Registration No.

26042

-Technical Report-

SPACIAL AND FREQUENCY DOMAIN CALCULATION OF TERRAIN ROUGHNESS METRIC ROOT-MEAN-SQUARE (RMS)

*****

*****

November 2014

U.S. Army Tank Automotive Research, Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

TARDEC Technical Report No. 26042
November 2014

SPACIAL AND FREQUENCY DOMAIN CALCULATION OF TERRAIN ROUGHNESS METRIC ROOT-MEAN-SQUARE (RMS)

W. Bylsma
Dynamics and Structures
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
Detroit Arsenal
ATTN: RDTA-SIE-SE/MS157
6501 East 11 Mile Road
Warren, Michigan 48397-5000

* * * * *

* * * * *

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
11-15-2014	TECHNICAL	2013-2014

SPACIAL AND FREQUENCY DOMAIN CALCULATION OF TERRAIN ROUGHNESS METRIC ROOT-MEAN-SQUARE (RMS)

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
WESLEY BYLSMA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DYNAMICS AND STRUCTURES-US ARMY RDECOM/TARDEC
ATTN: RDTA-SIE-SE/MS157
6501 E 11 MILE RD
WARREN, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

26042

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report documents the metric of root-mean-square (RMS) as a measure of terrain roughness--- recognizing its limitations (assumes no "favored or predominate frequencies") and accepting its historical significance to this point as the Army's measure of terrain roughness. Based on mathematics of the Fourier transform and duality, calculations are performed in the spacial and frequency domains to show equivalent measures of terrain roughness metric RMS.

15. SUBJECT TERMS
terrain roughness, RMS, spacial, frequency, duality

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	Unclassified	57	Wesley Bylsma
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(include area code)* 586-282-4104

CONTENTS

FIGURES

TABLES

SPACIAL AND FREQUENCY DOMAIN CALCULATION OF TERRAIN ROUGHNESS METRIC ROOT-MEAN-SQUARE (RMS)

TARDEC Technical Report No. 26042
November 2014

1.0 INTRODUCTION

Terrain roughness has a direct impact on ground vehicle performance. Movement over terrain causes excitation of the vehicle, through its suspension system, based on forward speed. Its effects can require reductions in speed due to vibration imparted to the occupants or concern about structural integrity. Performance requirements that incorporate terrain roughness, as a parameter, ensure vehicle operation at expected levels since the vehicle will be designed to handle excitations for that environment. Accomplishment of this requires a metric that measures terrain roughness which allows vehicle system designers and testers the ability to use it to correlate roughness levels with associated predicted and verified performance. As can be seen from references [1] to [11] this has been a field of interest for some time.

The scope here is limited to the metric of root-mean-square (RMS) as a measure of terrain roughness---recognizing its limitations (assumes no “favored or predominate frequencies”) and accepting its historical significance to this point as the Army’s measure of terrain roughness.

2.0 SPACIAL DOMAIN

The description and examples of the metric are given using MATLAB [13] as the computational platform.

2.1 Root-Mean-Square (RMS) Terrain Roughness Metric

The statistical measure of standard deviation (equivalent to the square root of the variance) “shows how much variation or dispersion from the average exists” (http://en.wikipedia.org/wiki/Standard_deviation). Another term for this is root-mean-square (RMS) when the mathematical operations used to calculate it are considered. The assumption that an average exists leads to the requirement that the metric must be used on stationary data to make sense. As Figure 1 shows, the RMS roughness metric for a non-stationary (non-detrended) terrain would result in a much larger value than for its stationary (detrended) counterpart and not provide the intended correlation between vehicle performance---the RMS “OF” the trend is much larger than the RMS of the profile “FROM” the trend.

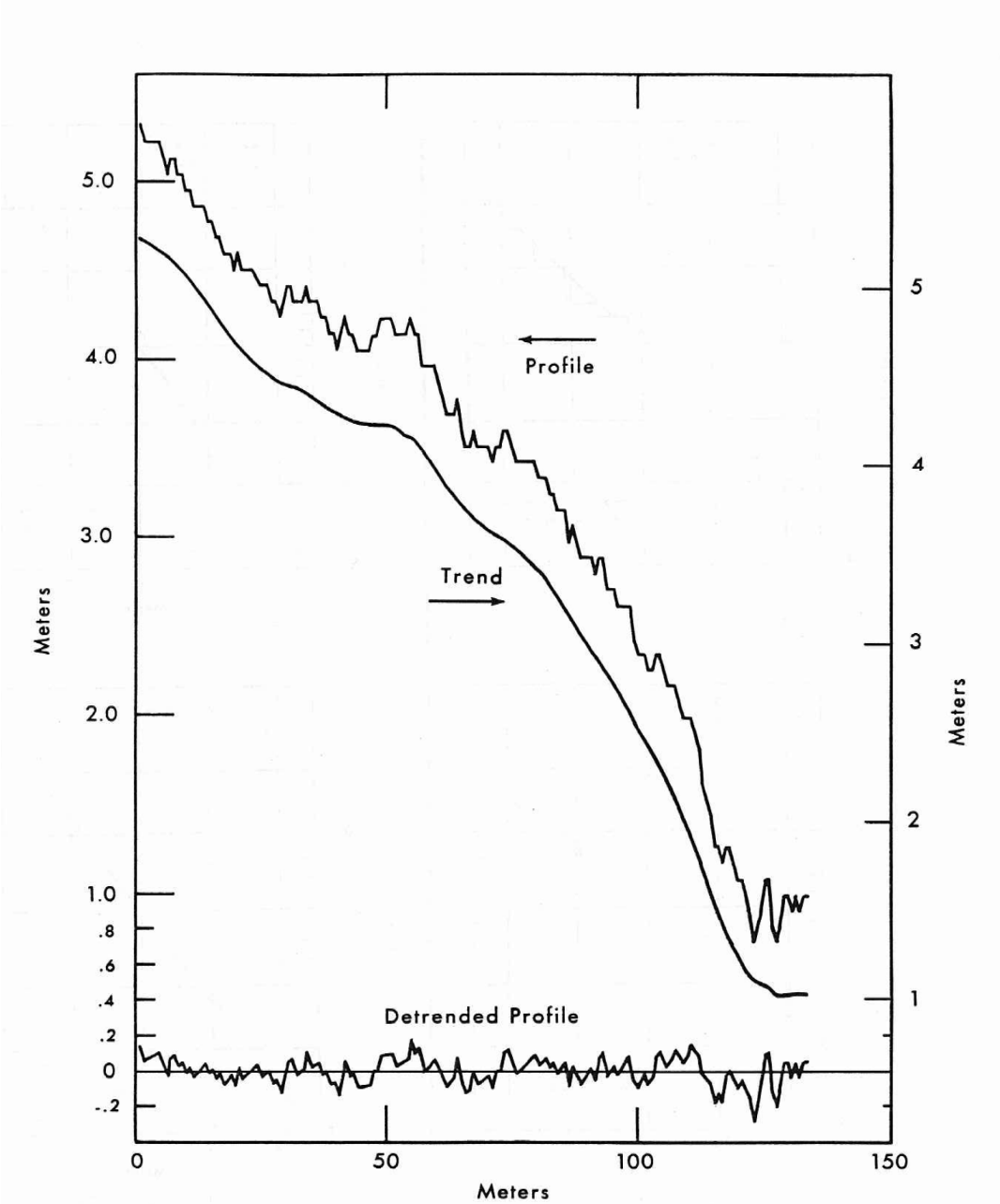


Figure A-9 EFFECT OF DETRENDING ON DATA SET 4

71

Figure 1 – Stationary Data Example (see [12])

This highlights the importance of detrending before calculating terrain roughness. Leaving the issue of which detrending method is best, further details of detrending with a double-sided exponential filter will be discussed in the formal definition of the RMS roughness metric.

Energy conservation indicates that there is a relationship between the spacial and frequency representation of a signal (see Parseval's theorem). The RMS roughness metric of a terrain profile

(spacial domain) is an equivalent measure of the area under the Power Spectral Density (PSD) of the same profile (frequency domain). From [12] the terrain PSD can be defined as

$$PSD_d(\lambda) = C\lambda^{-N} \tag{1}$$

where "N is approximately 2 for both natural and man-made surfaces. Man-made surfaces can be artificially constructed to give any value of N". With the definition in equation (2)

$$\lambda = \frac{v}{f} \tag{2}$$

λ	wavelength (distance/cycle)
v	velocity (distance/time)
f	frequency (cycle/time)

the dependence of the excitation of the vehicle on forward velocity is shown in equation (3).

$$PSD_d(f) = vCf^{-2} \tag{3}$$

From the above discussion, the formal definition of the RMS terrain roughness metric is given in Figure 2.

$$RMS = \sqrt{\frac{\sum_{i=0}^{N} F_d^2(x_i)}{N}}$$

$$F_d(x_i) = F(x_i) - \overline{F(x_i)}$$

$$\overline{F(x_i)} = \frac{\sum_{n=0}^{M}[F(x_n + na) + F(x_n - na)]e^{-\left(\frac{na}{\lambda}\right)}}{2\sum_{n=0}^{M} e^{-\left(\frac{na}{\lambda}\right)}}$$

$F(x)$	elevation at point x
$F_d(x)$	detrended elevation point
$\overline{F(x)}$	trend elevation point
n	step number
a	measurement interval
λ	weight constant

M	limit of summation

Figure 2 – RMS Terrain Roughness Metric Definition (see [12])

Application of the metric to a data sequence is simple. Making the data stationary (detrending) to obtain meaningful results is the difficulty---

> The complexity arises when the principles are reduced to practice, for then the realities of implementing logic, storing information, and transferring data have to be addressed.
>
> ---Robert F. Stengel, Optimal Control and Estimation (ISBN 0-486-68200-5)

2.2 Double-sided Exponential Filter

Based on work done in [12], a double-sided exponential filter has historically been used to detrend terrain elevation profile data before determining its RMS value---the necessity of which has been discussed above. In electrical engineering terms this is equivalent to high-pass filtering with the difference that for digital data double-sided averaging (both past and future) can be performed to remove phase shift that may result from one-sided filtering.

From Figure 2 values for the limit of summation, measurement interval, and weight constant must be determined to apply the detrending filter as shown in Figure 3.

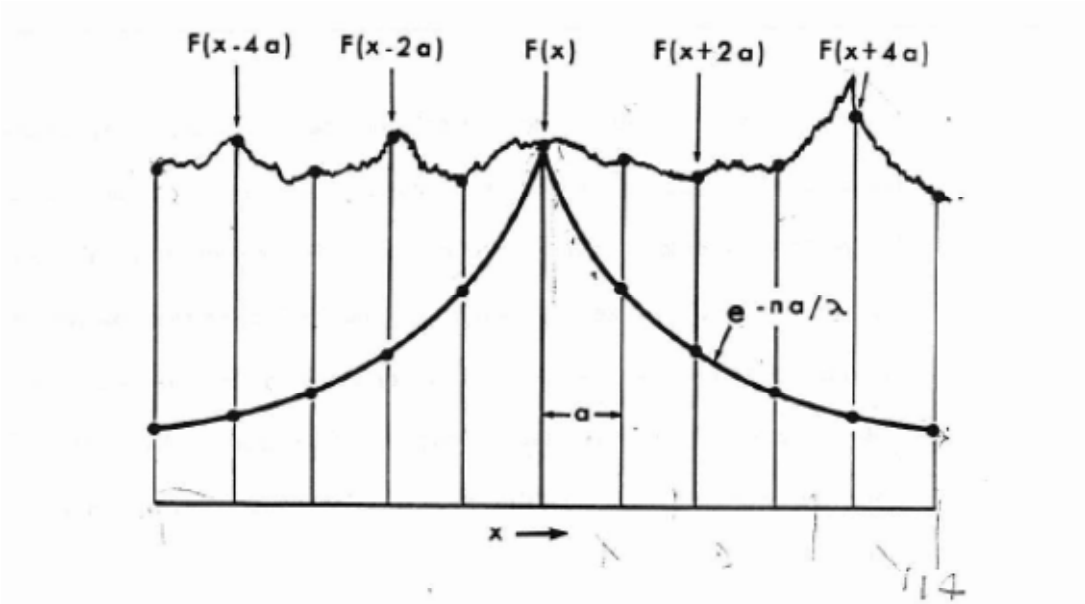


Figure A-2 EXPONENTIAL DETRENDING OF DATA POINTS

Figure 3 – Double-sided Exponential Filter (see [12])

2.2.1 Measurement Interval

Conclusions based on analysis in [12] indicate that the smallest wavelength that affects vehicles from natural terrains is about 2 feet. With consideration for the Nyquist–Shannon sampling theorem a maximum of 1 foot intervals can be chosen.

2.2.2 Limit of Summation

Historically, sixty feet (60 feet) has been considered the longest wavelength that affects vehicle responses. Equation (4) provides an example assuming the lowest sprung mass natural frequency is 0.75 Hz with a speed of 30 MPH (44 feet/sec).

$$\lambda_{\text{total}} = \frac{\text{v}}{\text{f}} = \frac{44}{0.75} = 58.6 \Rightarrow 60 \text{ feet}$$

(4)

This is the complete length of the double-sided exponential filter---halving the value for one-side (30 feet).

2.2.3 Weight Constant

As shown in equation (5) the practical numerical limit of the exponential filter's extent is considered met when the exponent reaches -3. Including terms with larger exponents will contribute little since their weighting coefficients will be quite small.

$$\text{e}^{-\left(\frac{\text{na}}{\lambda}\right)} = \text{e}^{-3} = 0.0497$$

(5)

Table 1 summaries the filter parameters for several cases.

Table 1 – Exponential Filter Parameters

exp length (1 side)	30 ft	360 in	360 in
a	1 ft	3 in	6 in
$M\left(=\frac{length}{a}\right)$	30	120	60
$\lambda\left(=\frac{Ma}{3}\right)$	10 ft	120 in	120 in

The extent is 3λ with λ being fixed for a given exponential length (1-side)---although the units may change (e.g. feet to inches).

2.3 Example Usage – RMS Terrain Roughness Metric Function

The RMS definition in Figure 2 is implemented in Appendix A.1 using MATLAB [13]. The function "rms" is used on digital elevation versus displacement samples in the spacial domain. The assumption for displacement is equally spaced data samples.

Example – calling rms function

```
% y hold digital elevation versus displacement data
x = y(:,2)*12; % convert feet to inches or consistent units for other input values
% e-(N*A/lambda)
N = [0:1:120]'; % 1-side = 30 ft = 360 in @ 3 in samples = 120 (360/3)
A = 3; % 3 in samples
lambda = 10*12; % 10 ft * 12 = 120 in
% -N*A/lambda = -120*3/120 = -3 @ last sample
[fdet,rms,fmean,xmean]=rms(x,N,A,lambda); % units are inches
```

The output of this function will be used to compare to frequency domain calculations.

3.0 FREQUENCY DOMAIN MATHEMATICS

Historically the RMS Terrain Roughness Metric has been calculated in the spacial domain. Digital Signal Processing (DSP) generally views filters from a frequency domain perspective. Addressing the frequency domain calculation of the metric in Section 2.1 and the filter defined in Section 2.2 should result in wider application and understanding.

The link between the spacial and frequency domains starts with the principle of duality between the two—multiplication in the time domain is convolution in the frequency domain, and vice versa. Other relationships between the two domains will briefly be discussed (Parseval's Theorem, etc.).

3.1 Correlation

Correlation is the first step in understanding convolution. An excellent reference is [15] from which the following definition is obtained and the first four pages are included in Appendix A.3.

$$F \circ I(x) = \sum_{i=-N}^{N} F(i) I(x+i)$$

Here F is considered of odd length. Consider it a sliding window operation---overlapping the sequences and then multiplying and adding. Additional coefficients may be involved---to perform functions such as averaging, etc. When F does not completely overlap I operations are undefined for values outside of the boundaries of I. Several options are available.

- pad with zeros (MATLAB default)
- pad with 1st/last values
- cyclically repeated

Computation can continue when undefined values are resolved with one of these methods.

3.2 Convolution

Convolution is the dual operation to multiplication in the spacial and frequency domains. It is the same as correlation, except that the filter is flipped first [15] (e.g. F=(2,8,9) -> (9,8,2)) and defined as

$$F * I(x) = \sum_{i=-N}^{N} F(i) I(x-i)$$

Another useful reference on convolution is [16].

3.3 Parseval's Theorem

One of the other relationships between the spacial and frequency domain is Parseval's Theorem [17]. It is defined as

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2$$

which shows the equivalence between energy in the signal regardless of the domain. It is used to check computational results to ensure the principle is preserved.

3.4 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is a mathematical tool [18] used to change data representations between the spacial and frequency domains. A variant that allows faster compute times is called the Fast Fourier Transform (FFT) but performs the equivalent function. The inverse DFT of convolution results in the frequency domain should match multiplication results in the spacial domain. This method will be used to verify frequency domain calculations.

3.4.1 Scaling

From [18] the following is informative regarding the scale factor applied to the DFT:

> The normalization factor multiplying the DFT and IDFT (here 1 and 1/N) and the signs of the exponents are merely conventions, and differ in some treatments. The only requirements of these conventions are that the DFT and IDFT have opposite-sign exponents and that the product of their normalization factors be 1/N. A normalization of $\sqrt{1/N}$ for both the DFT and IDFT, for instance, makes the transforms unitary.

Note that MATLAB uses the opposite scaling mentioned in the first sentence. From [19] elaboration on scaling is given:

> Question 3.7. FFTW gives results different from my old FFT.
>
> People follow many different conventions for the DFT, and you should be sure to know the ones that we use (described in the FFTW manual). In particular, you should be aware that the FFTW_FORWARD/FFTW_BACKWARD directions correspond to signs of -1/+1 in the exponent of the DFT definition. (Numerical Recipes uses the opposite convention.). You should also know that we compute an unnormalized transform. In contrast, Matlab is an example of program that computes a normalized transform. See Q3.10 `Why does your inverse transform return a scaled result?'.
>
> Finally, note that floating-point arithmetic is not exact, so different FFT algorithms will give slightly different results (on the order of the numerical accuracy; typically a fractional difference of 1e-15 or so in double precision).
>
> ...
>
> Question 3.10. Why does your inverse transform return a scaled result?
>
> Computing the forward transform followed by the backward transform (or vice versa) yields the original array scaled by the size of the array. (For multi-dimensional transforms, the size of the array is the product of the dimensions.) We could, instead, have chosen a normalization that would have returned the unscaled array. Or, to accommodate the many conventions in this matter, the transform routines could have accepted a "scale factor" parameter. We did not do this, however, for two reasons. First, we didn't want to sacrifice performance in the common case where the scale factor is 1. Second, in real applications the FFT is followed or preceded by some computation on the data, into which the scale factor can typically be absorbed at little or no cost.

3.5 Fast Fourier Transform (FFT) Proper Scaling Test

Figure 4 shows the proper scaling of the FFT in MATLAB to match the input signal. From the three term sinusoidal combination the frequency domain peak values match those defined in the time domain signal.

```
% MATLAB example
Fs1 = 1000;    % F=1/T =1/1e-3sec = 1/msec
L = 1000;
t= (0:1/Fs1:5000-1);    % 5 sec @ Fs=1000
y = 0.3*sin(2*pi*35*t)+0.7 *sin(2*pi*50*t) + sin(2*pi*120*t);
tNFFT = 2^nextpow2(L);
tY = fft(y,tNFFT)/tNFFT;
tf = Fs1/2*linspace(0,1,tNFFT/2+1);    % one sided frequency

subplot(2,1,1),plot(1000*t(1:250),y(1:250))
subplot(2,1,2),plot(tf,2*abs(tY(1:tNFFT/2+1)))
```

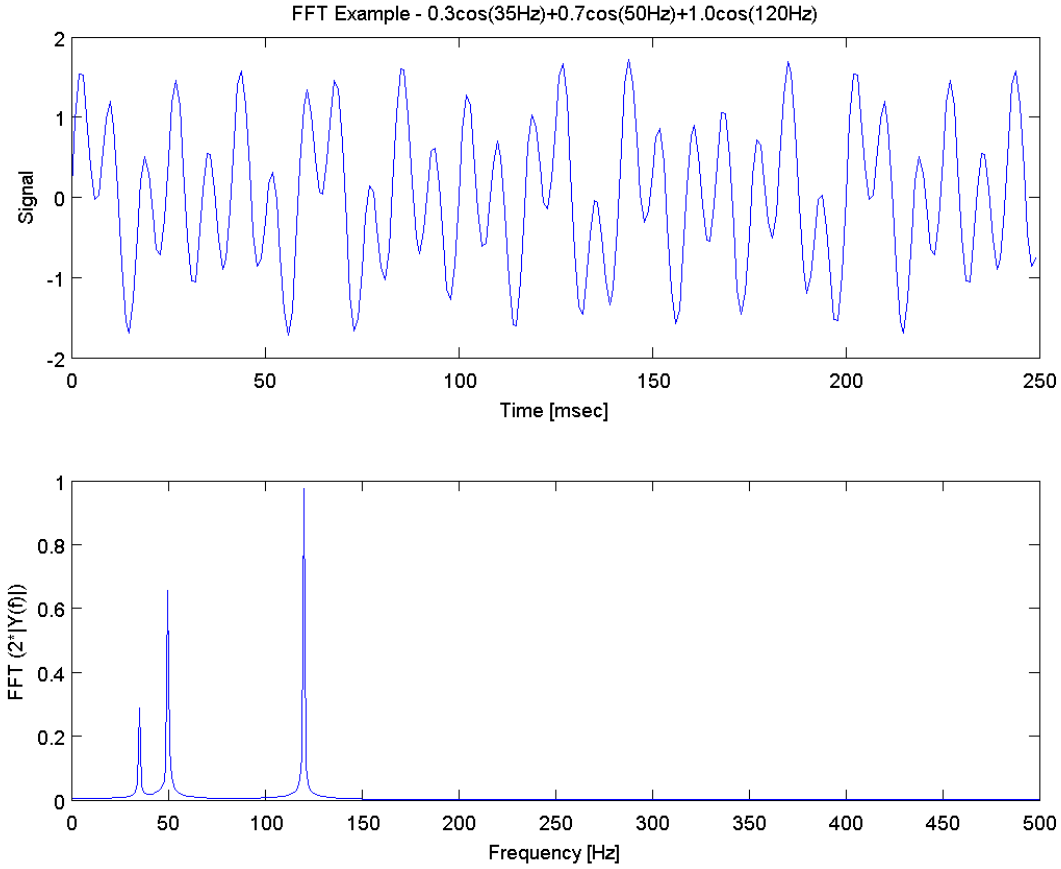


Figure 4 – FFT Example with Proper Scaling

4.0 FREQUENCY DOMAIN

Application of Section 2.0 using Section 3.0 will be demonstrated in Section 4.0.

4.1 Root-Mean-Square (RMS) Terrain Roughness Metric

The calculation of the RMS terrain roughness metric in the frequency domain is based on the duality of convolution (multiplication in the spacial domain). The approach here is to provide equivalent frequency domain operations and examples that show the same results as in the spacial domain with a plot of results and description of the MATLAB script used to generate it. It should be noted that index values in MATLAB are one (1) based and not zero (0) based.

4.2 Munson Gravel

The following examples apply to the profile known as Munson Gravel.

4.2.1 Spacial Domain

As described in Section 2.3 the time domain is the starting point. Figure 5 is based on time domain operations (see Appendix A.1). Notice that the mean and detrended profile lose one half the length of the filter at the beginning and end.

```
y=dlmread('MUNGL.FIL');    % read in the data
x = y(:,2)*12;       % convert feet to inches or consistent units

tmp = size(x);
lenx = tmp(1);    % get the length (# of points)

% e-(N*A/lambda)
N = [0:1:120]';      % 1-side = 30 ft = 360 in @ 3 in samples = 120 (360/3)
A = 3;      % 3 in samples
lambda = 10*12;      % 10 ft * 12 = 120 in
% -N*A/lambda = -120*3/120 = -3 @ last sample
[fdet,rms,fmean,xmean]=rmswes(x,N,A,lambda);

tt1 = (0:lenx-1)'*(1/3);      % displacement, 3 in samples
plot(tt1,x,'r');
hold on
plot(tt1(121:end-120),fmean,'g');      % account for filter beg and end
plot(tt1(121:end-120),fdet,'b');       % account for filter beg and end
```

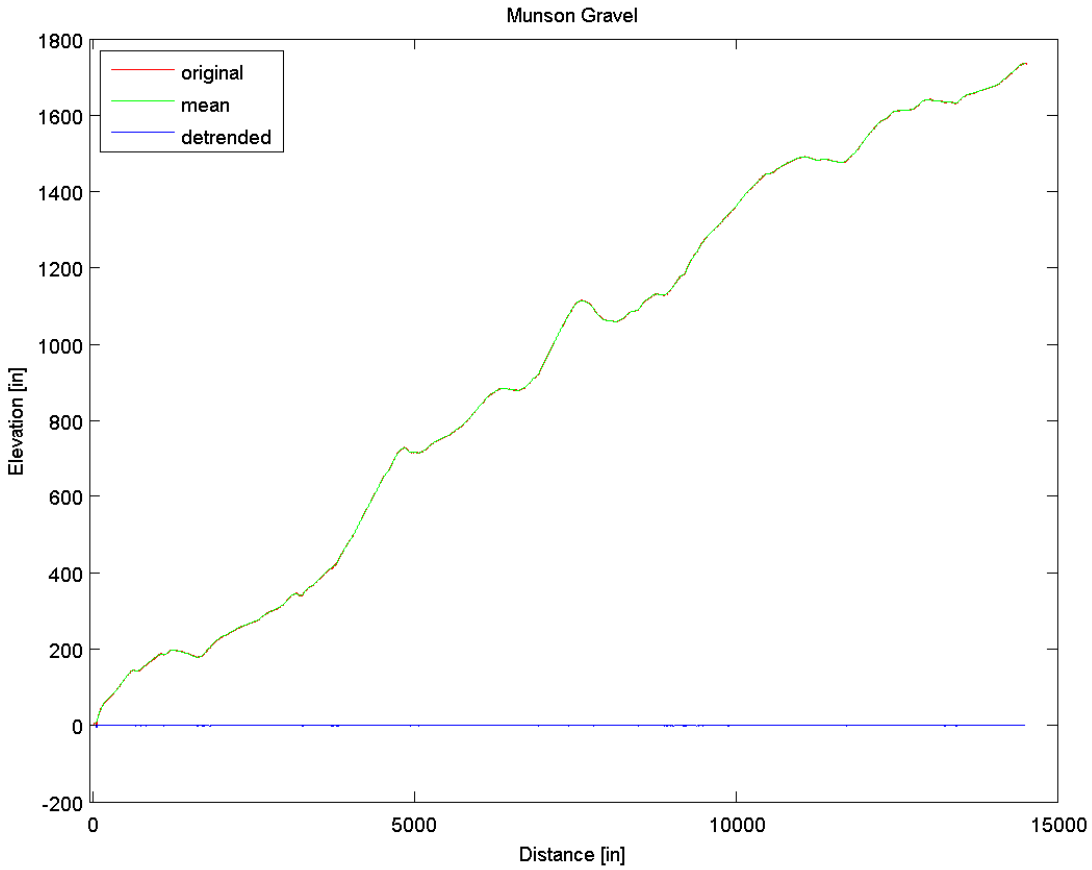


Figure 5 – Spacial Domain RMS Terrain Roughness Metric Results

Figure 6 shows a zoomed view of Figure 5 for the terrain profile and trend based on the filter defined in Section 2.2. Notice the missing samples in the trend if the original data is not extended by the length of the filter to avoid loss of data.

```
...
plot(tt1,x,'r');
hold on
plot(tt1(121:end-120),fmean,'g');
axis([1.444e4,1.455e4,1732,1737])
```

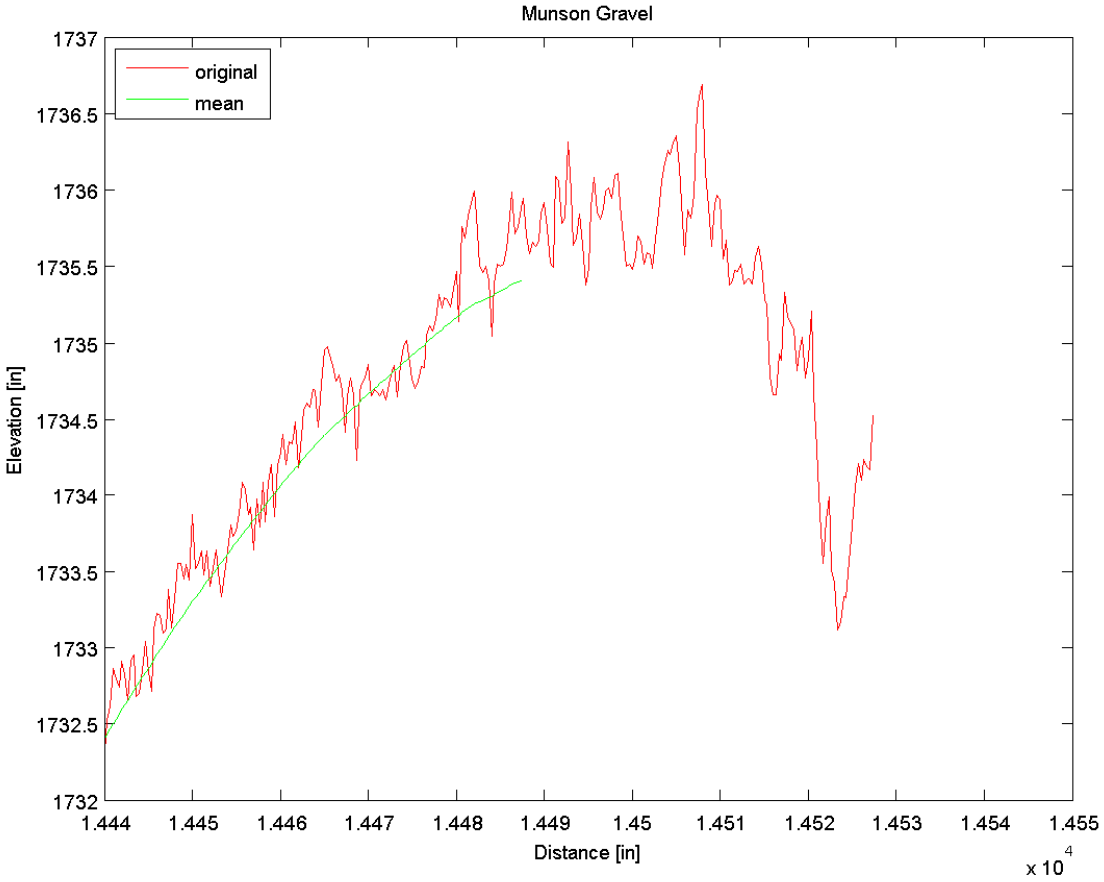


Figure 6 – Zoomed Terrain Profile and Trend

Figure 7 shows a zoomed view of the detrended terrain profile.

```
...
plot(tt1,x,'r');
hold on
plot(tt1(121:end-120),fmean,'g');
axis([1.444e4,1.455e4,1732,1737])
```

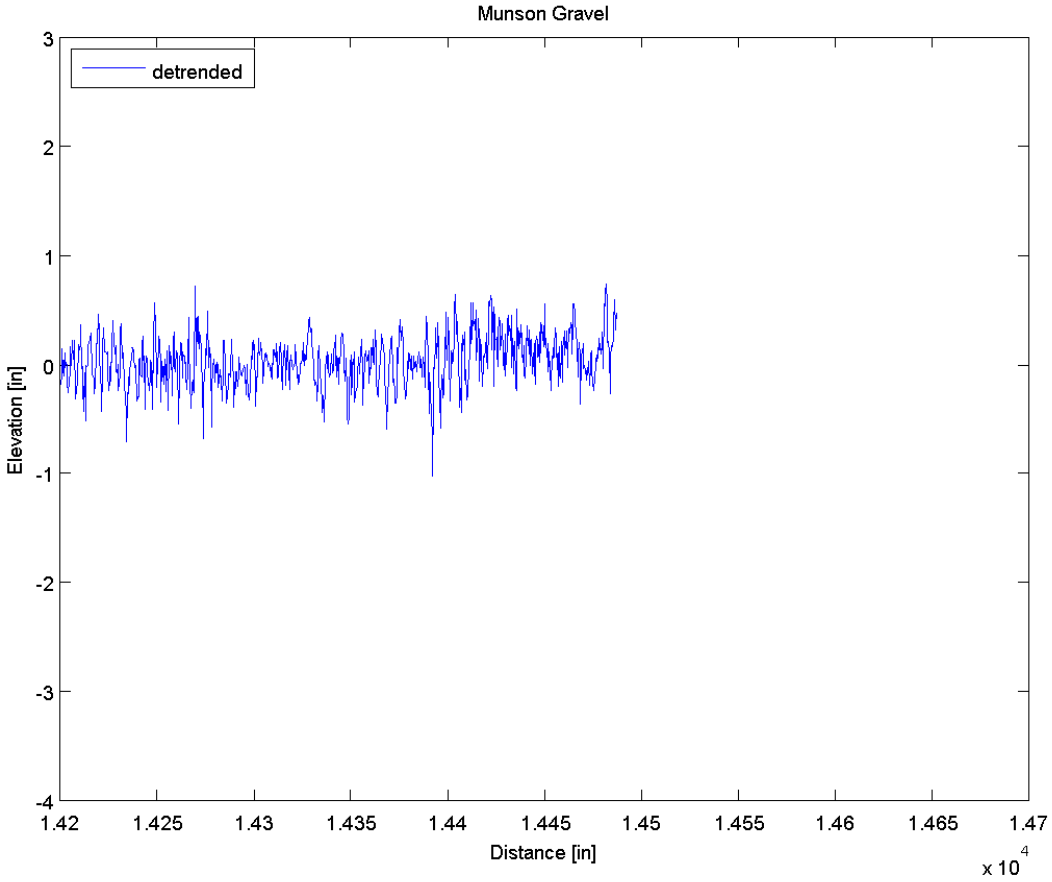


Figure 7 – Detrended Terrain Profile

4.2.2 Spacial Domain Filters

Figure 8 shows normalized 2-sided exponential and rectangular filters. Notice the overlap at fexp(lenN).

```
...
% 2-sided exponential
tmp = size(N);
lenN = tmp(1);
fexp = zeros(lenN*2-1,1);     % overlap at zero
fexp(1:lenN) = exp(-N(length(N):-1:1)*A/lambda);
fexp(lenN:end) = exp(-N*A/lambda);
fexp(lenN) = 2;      % double count zero position
fexp = fexp/sum(fexp);
tmp = size(fexp);
lenfexp = tmp(1);

% rect
rect = ones(lenN*2-1,1);
rect = rect/sum(rect);
tmp = size(rect);
lenrect = tmp(1);

subplot(2,1,1),plot(fexp,'r')
subplot(2,1,2),plot(rect,'g')
```

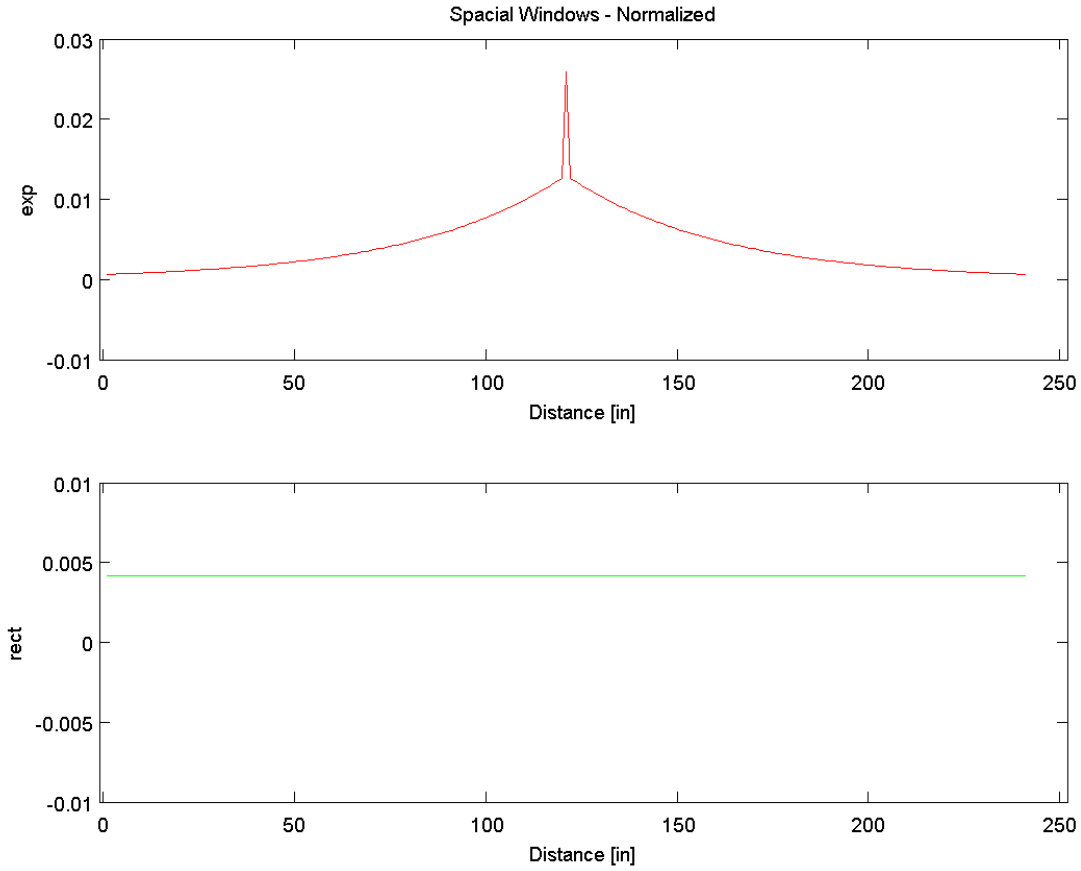


Figure 8 – Spacial Domain Filters

4.2.3 Frequency Domain Filters

Figure 9 shows the frequency domain version of the spacial domain filters. In real frequency terms, accounting for negative frequencies, only one half of the length (NFFT) is used with double the absolute value of the magnitude.

```
...
% filters in frequency domain - FFT
NFFT = 2^nextpow2(lenx);      % !!!use all with this so can mul in freq domain

Ffexp = fft(fexp,NFFT)/NFFT;  % !!!scale by len to get right mag (see sin test)
Frect = fft(rect,NFFT)/NFFT;
% Fs = 1/T
Fs=(1/3); % 4 samples in 12 in = 4/12 = 1/3

% linspace(x1,x2,N) = N pts between x1 and x2
f = Fs/2*linspace(0,1,NFFT/2+1);

subplot(2,1,1),plot(f,2*abs(Ffexp(1:NFFT/2+1)),'r');     % !!! 2*abs /NFFT/2+1
subplot(2,1,2),plot(f,2*abs(Frect(1:NFFT/2+1)),'g');
```

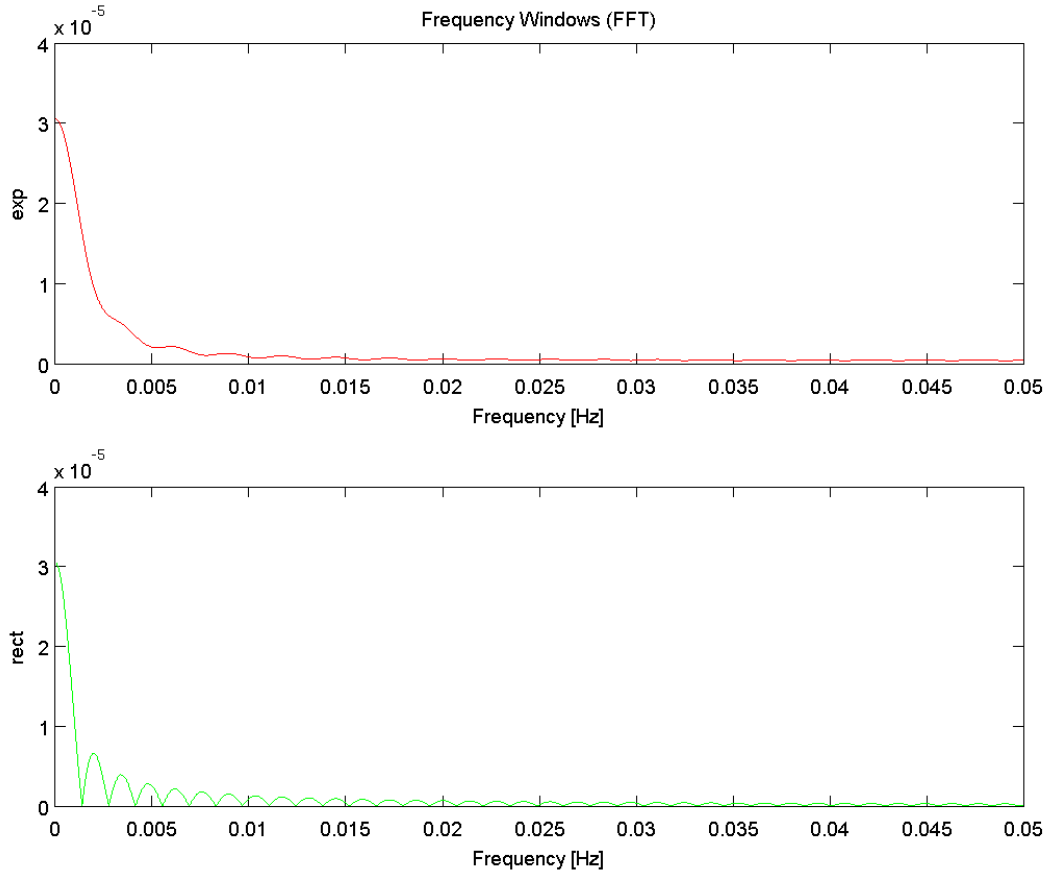


Figure 9 – Frequency Domain Filters

4.2.3.1 Spacial Domain vs. Frequency Domain Filter Differences

Based on proper scaling demonstrated in Section 3.5, differences in the spacial domain and inverse FFT of the frequency domain defined filters are shown in Figure 10. Note they are the same except for round off error.

```
...
% filter differences - FFT/inverse FFT
ffexp = NFFT*ifft(Ffexp,NFFT);      %!!! scale by length
frect = NFFT*ifft(Frect,NFFT);

subplot(2,1,1),plot(fexp-ffexp(1:lenfexp),'r');    % lenfexp = lenN*2-1
subplot(2,1,2),plot(rect-frect(1:lenrect),'b');
```

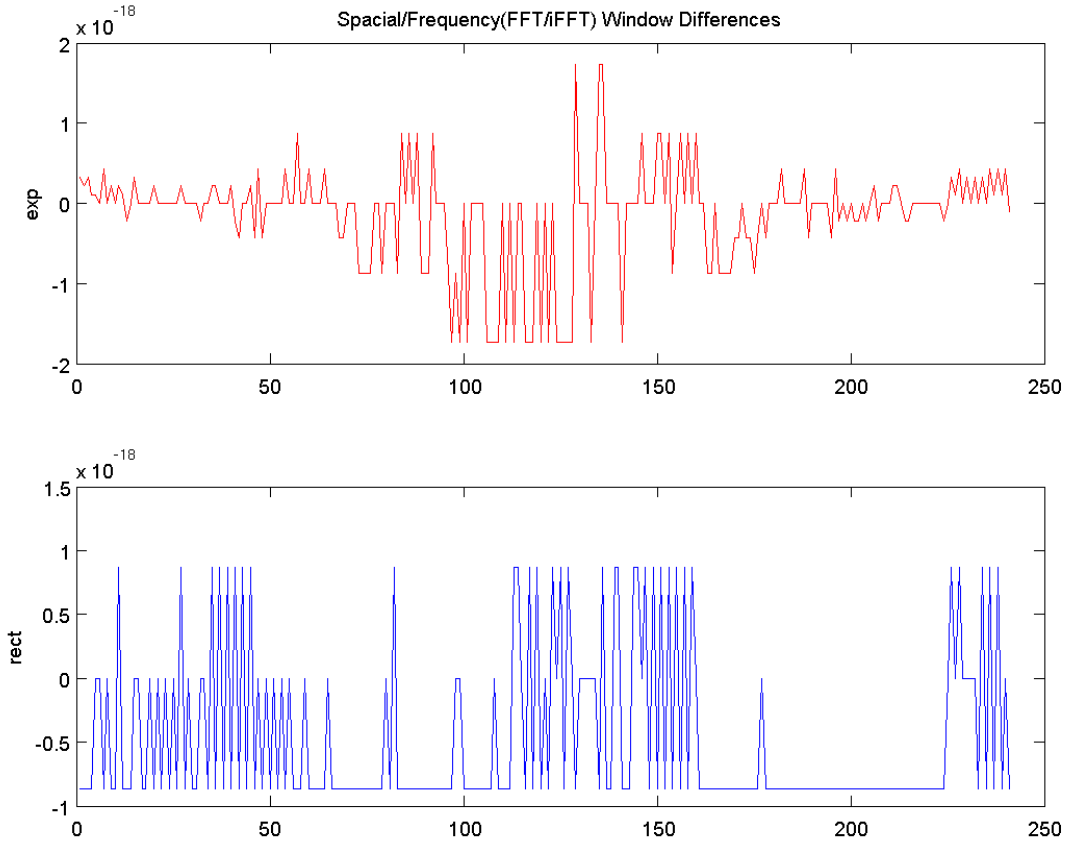


Figure 10 – Spacial versus Frequency Domain Filter Differences

4.2.4 Frequency Domain Operations – FFT/iFFT

Figure 11 shows the error between the original terrain profile and the FFT/iFFT of it---indicating the operation gives back the original with no error.

```
...
% FFT x
Fx = fft(x,NFFT)/NFFT;
tt6=(0:lenx-1)*(1/3);
ft6 = Fs/2*linspace(0,1,NFFT/2+1);

% inverse FFT x
Fcx = Fx;
fFcx = NFFT*ifft(Fcx,NFFT);

plot(tt6,x,'r');
hold on
plot(tt6,fFcx(1:lenx),'g*'); % !!! NFFT -> true length
plot(tt6,x-fFcx(1:lenx),'b');
```

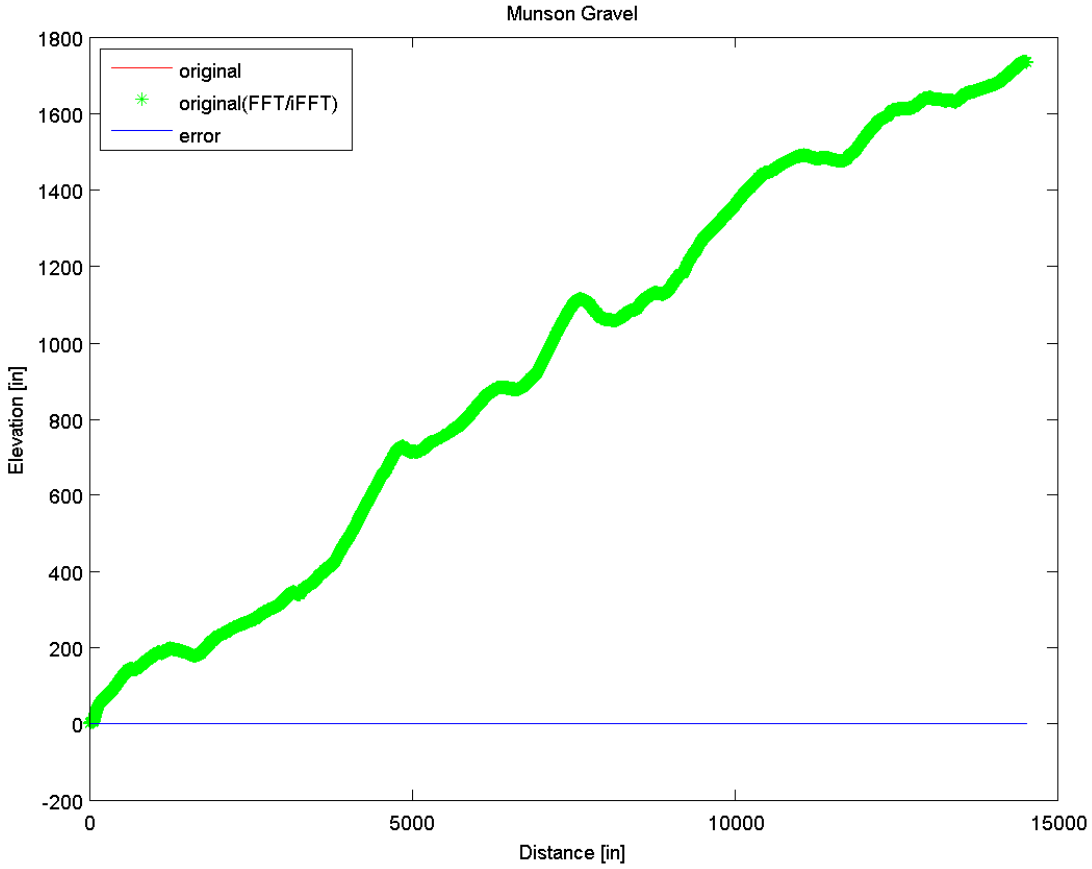


Figure 11 – Frequency Domain Operations - FFT/iFFT Comparison

Figure 12 zooms in on the end of the original and FFT/iFFT terrain profile to show both are equivalent, validating the representation of the terrain profile in the spacial or frequency domain.

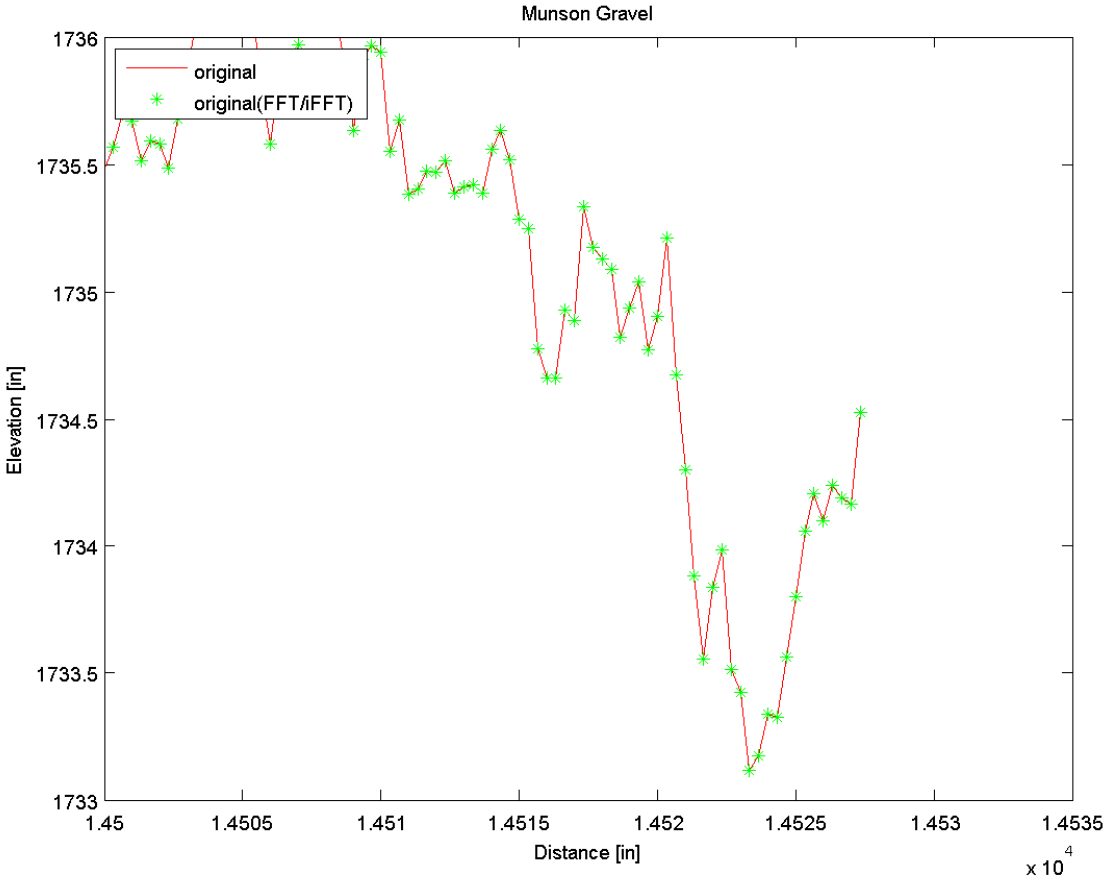


Figure 12 – Zoomed FFT/iFFT Comparison

4.2.5 Spacial vs. Frequency Domain Mean Comparison

Figure 13 shows the comparison of the terrain profile trend (or mean) where it was filtered in the spacial and frequency domain showing the equivalence of multiplication in the spacial domain and convolution in the frequency domain.

```
...
% exp filter
H = Ffexp;
tmp = size(H);
lenH = tmp(1);

% inverse x, mean, mydet, detrended
Fcx = Fx.*H;
fFcx = NFFT*NFFT*ifft(Fcx,NFFT);    % !!! scale by time length

plot(fmean,'r')
hold on
% !!! proper indexes for y cutoffs (y only when exp fully in orig y)
plot(abs(fFcx(lenfexp:lenx+(lenfexp-1)-(lenfexp-1))),'g*');
```

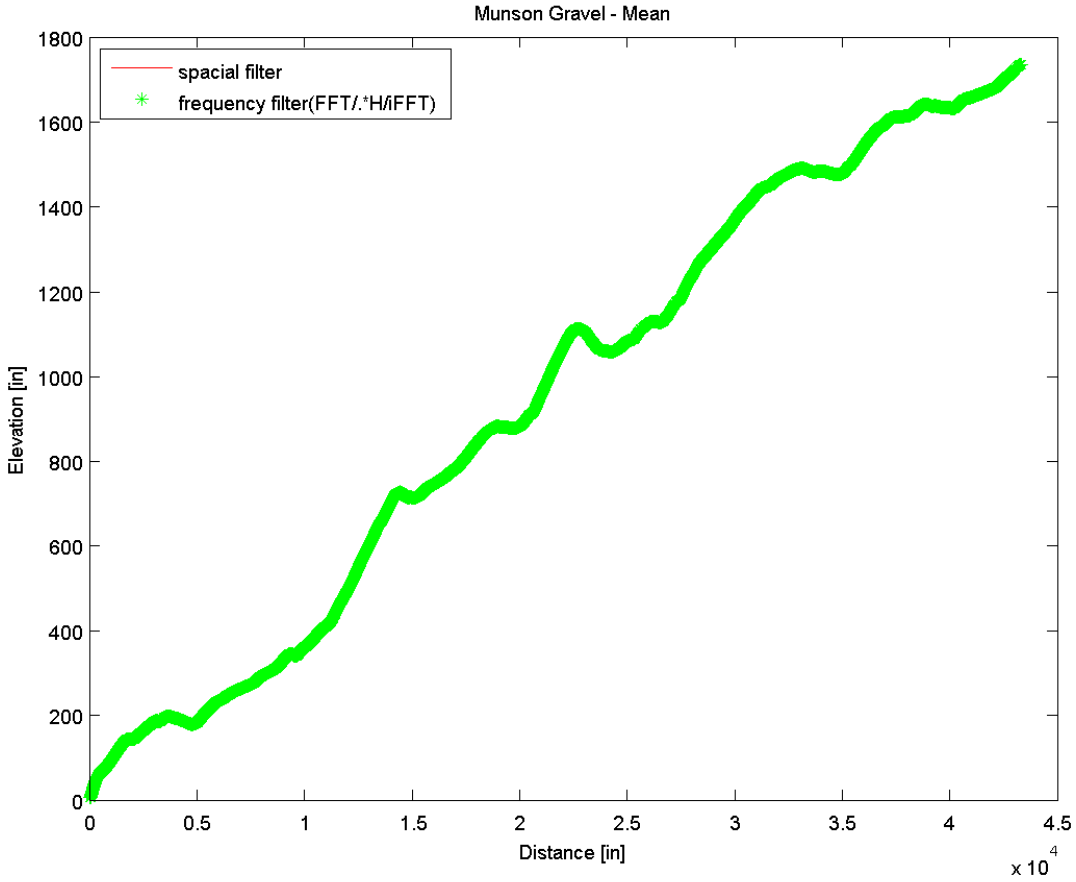


Figure 13 – Spacial versus Frequency Domain Mean Comparison

A zoomed version is shown in Figure 14 for the end of the terrain profile.

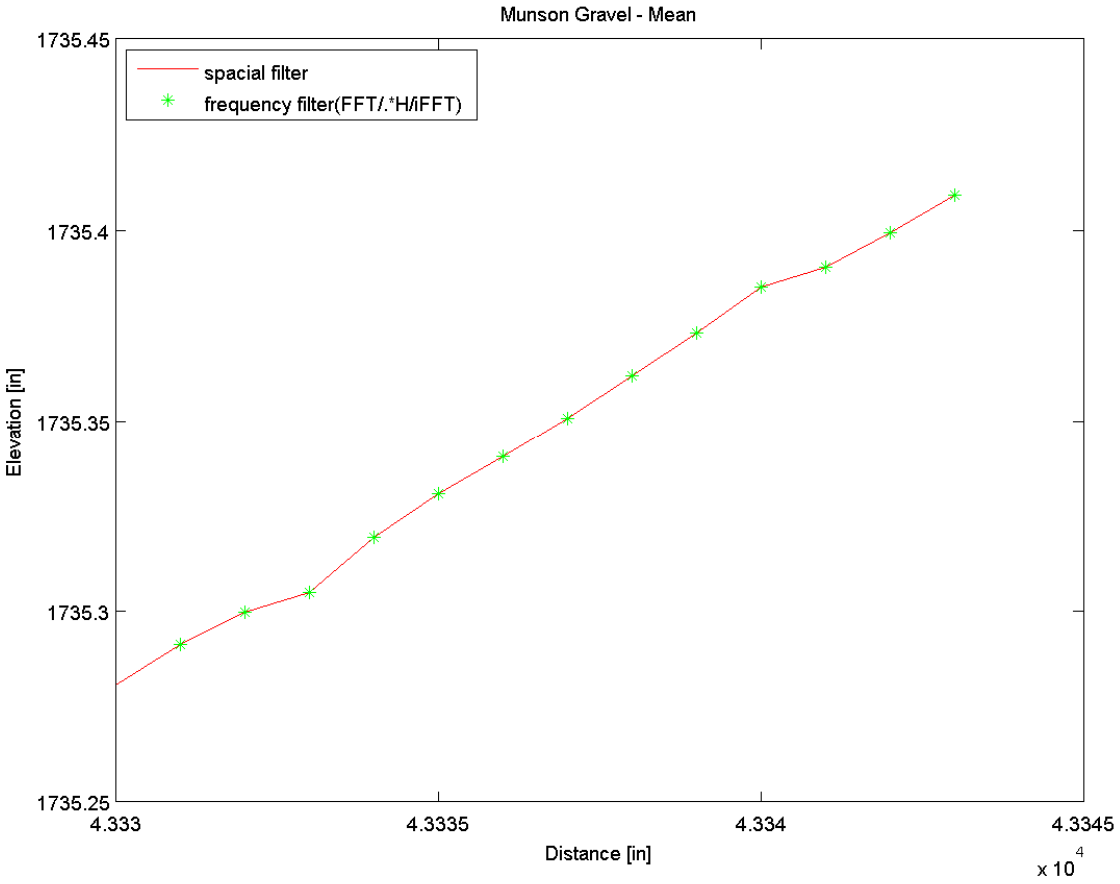


Figure 14 – Zoomed Mean Comparison

4.2.6 Spacial vs. Frequency Domain Detrended Comparison

Figure 15 shows the detrended terrain profile based on subtracting the trend obtained from the frequency domain.

```
...
% detrended
% assumes lenexp is odd, take out mean from FFT/.*H/iFFT
mydet = x( (lenfexp+1)/2 : lenx - (lenfexp+1)/2 + 1 ) - abs( fFcx( lenfexp : lenx + (lenfexp-
1) - (lenfexp-1) ) );
mm=mean(mydet)
mydet = mydet - mm;

plot(fdet,'r');
hold on
plot( mydet,'g*')
```

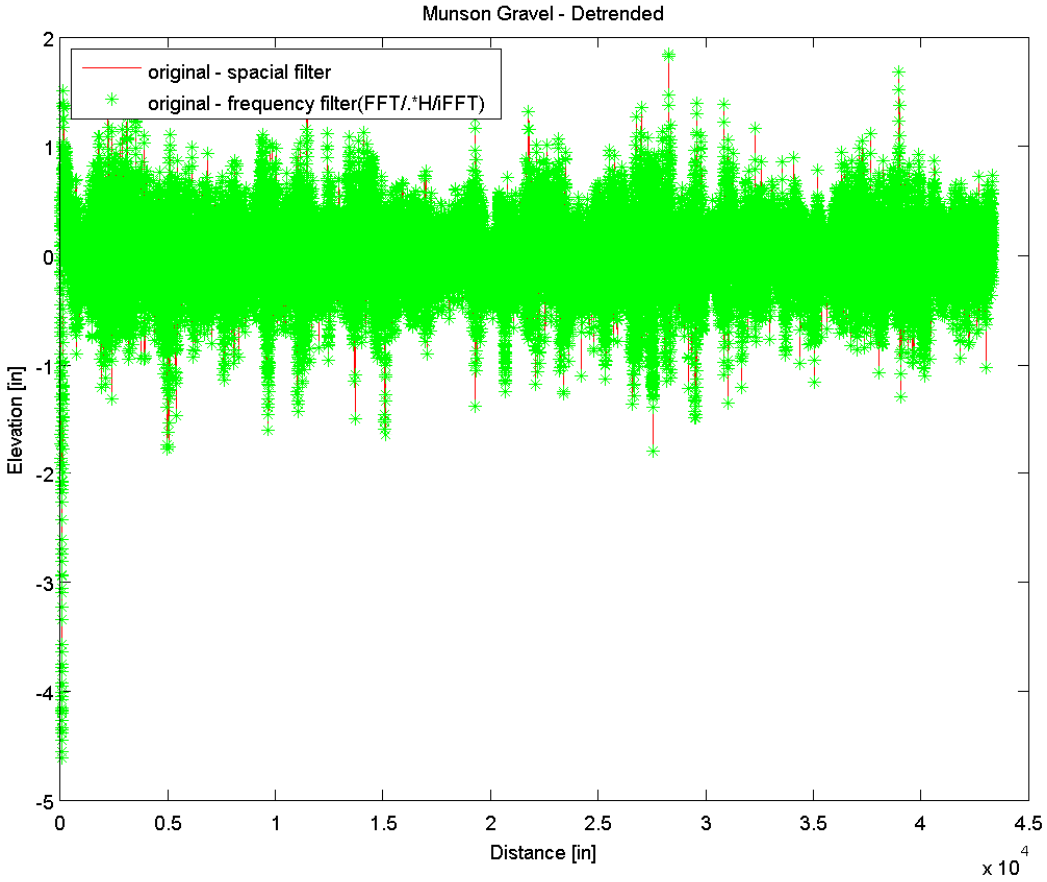


Figure 15 – Spacial versus Frequency Domain Detrended Comparison

A zoomed version of the end is shown in Figure 16. Note the agreement in results.

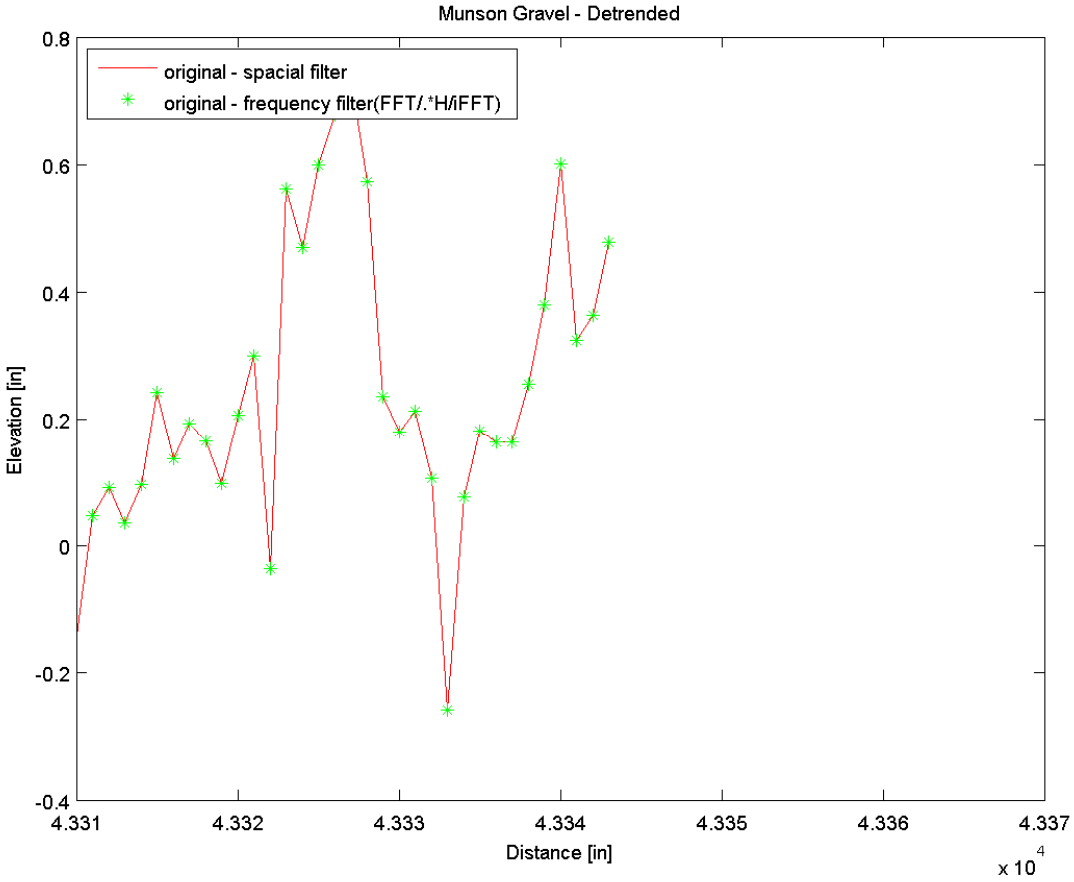


Figure 16 – Zoomed Detrended Comparison

4.2.7 Spacial vs. Frequency Domain Mean and Detrended Differences

Figure 17 shows the difference between the spacial and frequency domain for the mean and detrended profile. Note they are the same except for round off error---about machine precision.

```
...
% mean error
% !!! proper indexes for y cutoffs (y only when exp fully in orig y)
subplot(2,1,1),plot(fmean - abs(fFcx(lenfexp:lenx+(lenfexp-1)-(lenfexp-1))),'r');

% detrended error
subplot(2,1,2),plot(fdet-mydet,'r');
```

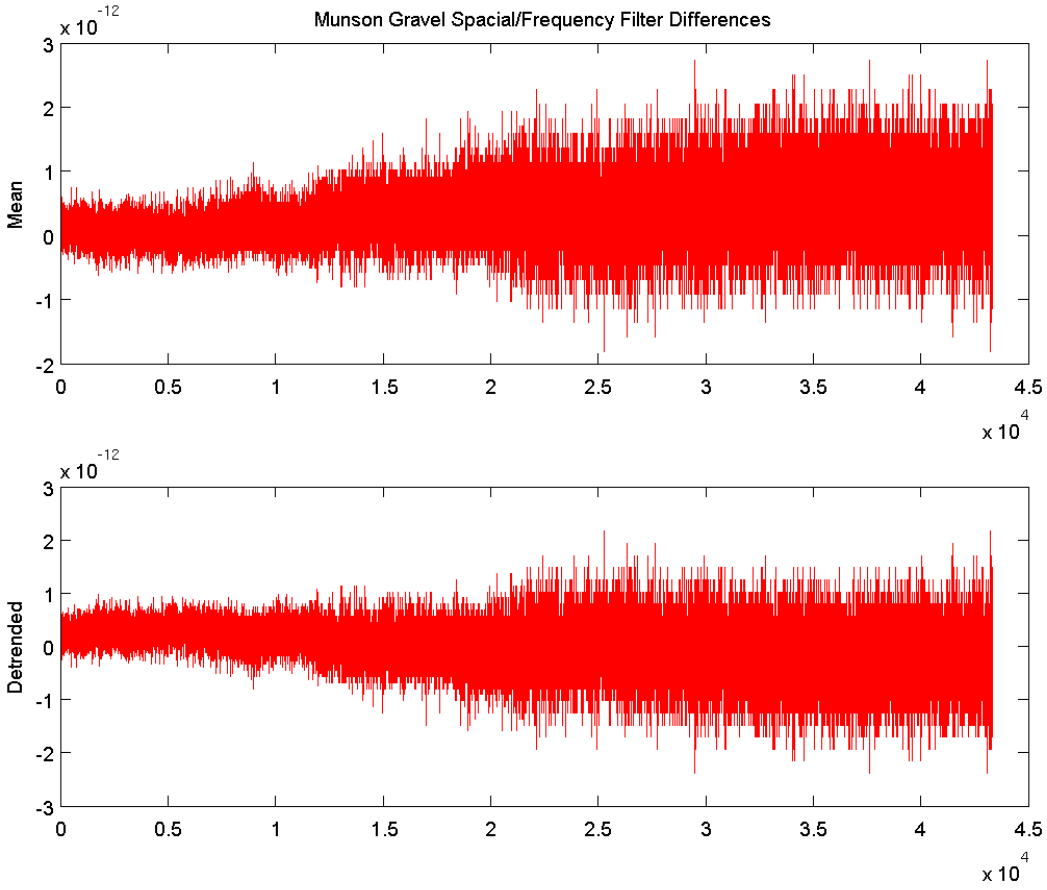


Figure 17 – Spacial versus Frequency Domain Mean/Detrended Differences

4.2.8 Equivalent Measures

Results output from the script below show numerical comparisons between the spacial domain RMS function, standard deviation of the detrended signal, square root of the variance of the detrended signal, and each side of Parseval's Theorem outlined in Section 3.3 to be equivalent.

```
...
% PSD
% std(x,1) = sqrt(sum(x.^2)/length(x))

tmp = size(mydet);
lenmydet = tmp(1)

Fmydet = fft(mydet,NFFT)/NFFT;
Pxx = Fmydet.*conj(Fmydet);

!rm t10-d
diary t10-d
'spacial/freq results'
'rms'
'std(mydet,1)'
'sqrt(var(mydet,1))'
'sqrt( sum(mydet.^2)/length(mydet))'
'sqrt( sum( Pxx )*length(mydet)/NFFT)'

[rms,std(mydet,1),sqrt(var(mydet,1)),sqrt(    sum(mydet.^2)/length(mydet)),sqrt(    sum(    Pxx
)*NFFT/length(mydet))]
diary
```

```
ans =

spacial/freq results


ans =

rms


ans =

std(mydet,1)


ans =

sqrt(var(mydet,1))


ans =

sqrt( sum(mydet.^2)/length(mydet))


ans =

sqrt( sum( Pxx )*length(mydet)/NFFT)


ans =

    0.3364    0.3364    0.3364    0.3364    0.3364
```

Computation of the Root-Mean-Square (RMS) Terrain Roughness Metric (Section 2.1) is valid in either the spacial or frequency domain.

4.3 Perryman 3

The following examples apply to the profile known as Perryman 3. Differences from Section 4.2 are only in the file read in and label names.

4.3.1 Spacial Domain

As described in Section 2.3 the time domain is the starting point. Figure 18 is based on time domain operations.

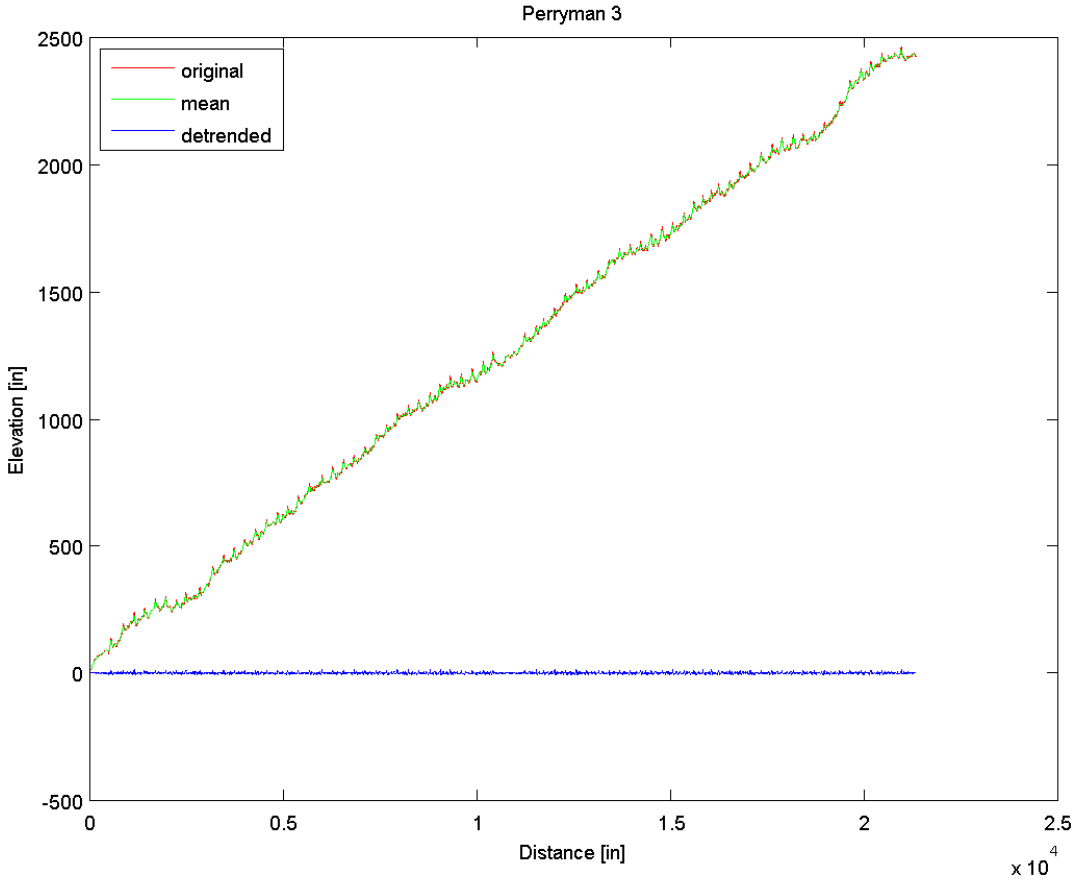


Figure 18 – Spacial Domain RMS Terrain Roughness Metric Results

Figure 19 shows a zoomed view of Figure 18 for the terrain profile and trend based on the filter defined in Section 2.2. Notice the missing samples in the trend if the original data is not extended by the length of the filter to avoid loss of data.

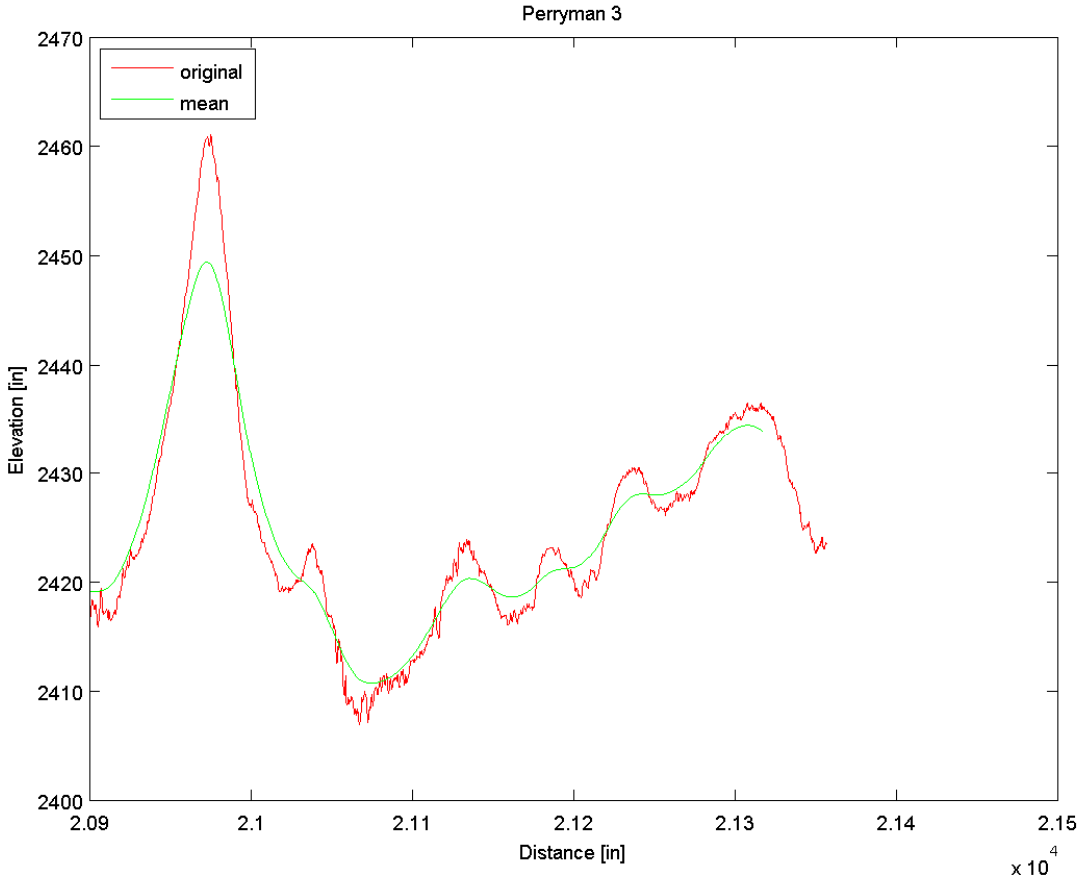


Figure 19 – Zoomed Terrain Profile and Trend

Figure 20 shows a zoomed view of the detrended terrain profile.

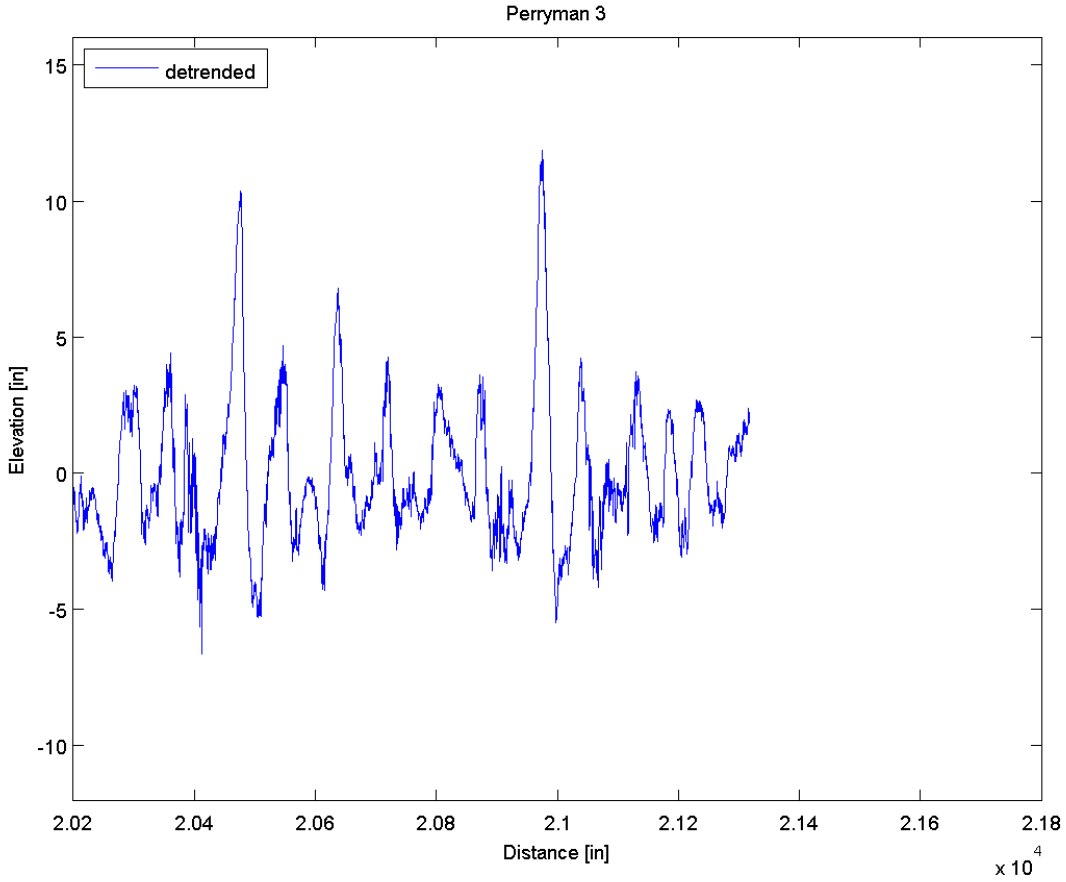


Figure 20 – Detrended Terrain Profile

4.3.2 Frequency Domain Operations - FFT/iFFT

Figure 21 shows the error between the original terrain profile and the FFT/iFFT of it---indicating the operation gives back the original with no error.

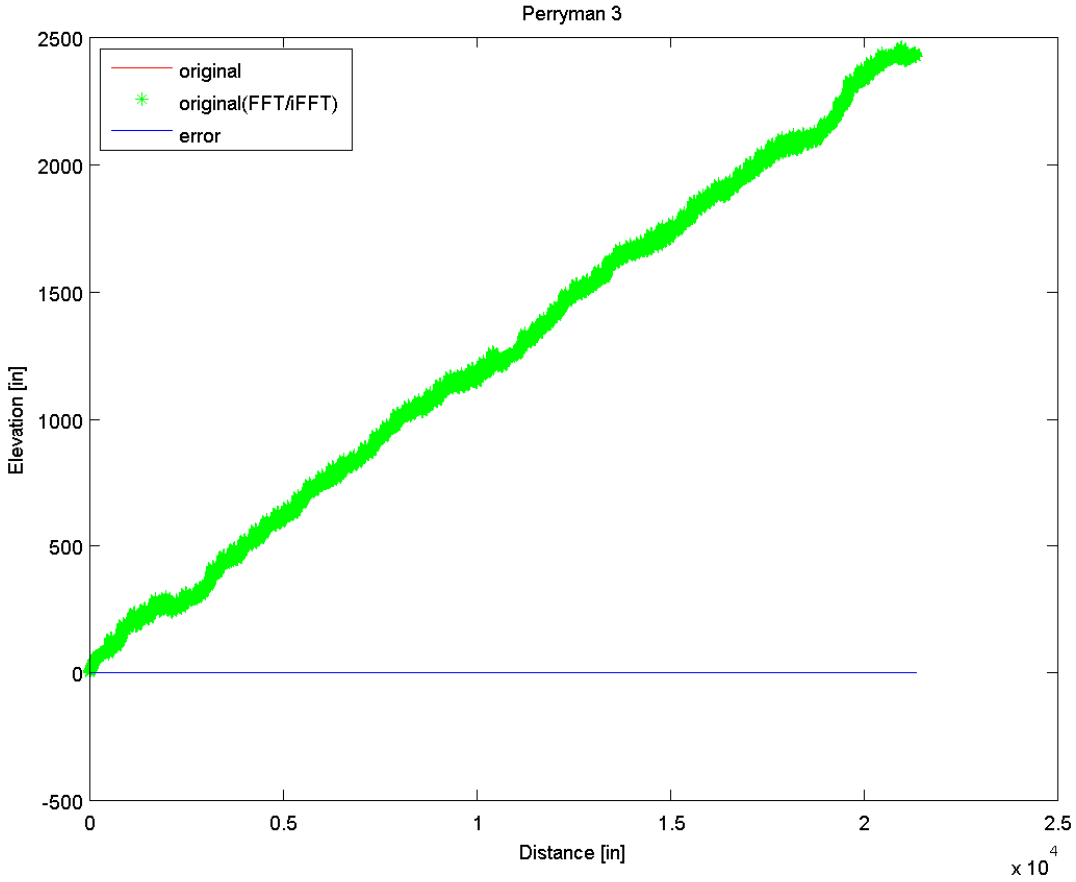


Figure 21 – Frequency Domain Operations - FFT/iFFT Comparison

Figure 22 zooms in on the end of the original and FFT/iFFT terrain profile to show both are equivalent, validating the representation of the terrain profile in the spacial or frequency domain.

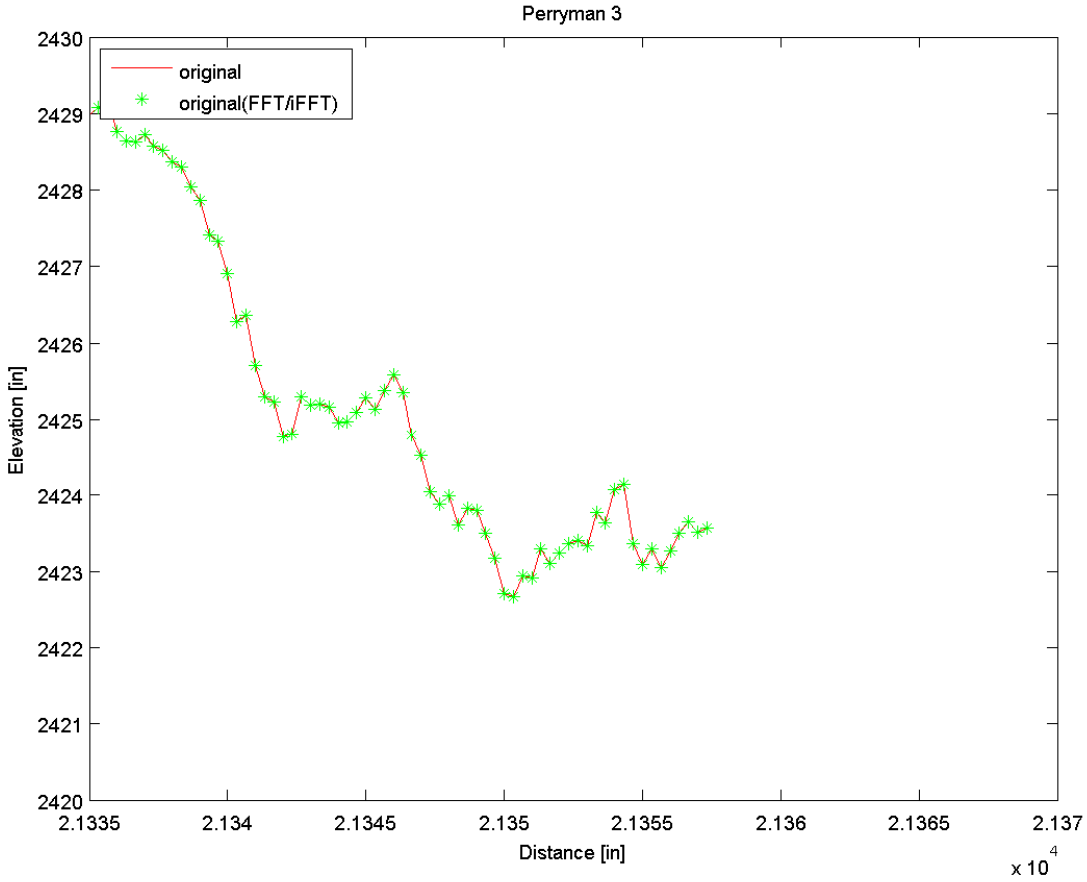


Figure 22 – Zoomed FFT/iFFT Comparison

4.3.3 Spacial vs. Frequency Domain Mean Comparison

Figure 23 shows the comparison of the terrain profile trend (or mean) where it was filtered in the spacial and frequency domain showing the equivalence of multiplication in the spacial domain and convolution in the frequency domain.

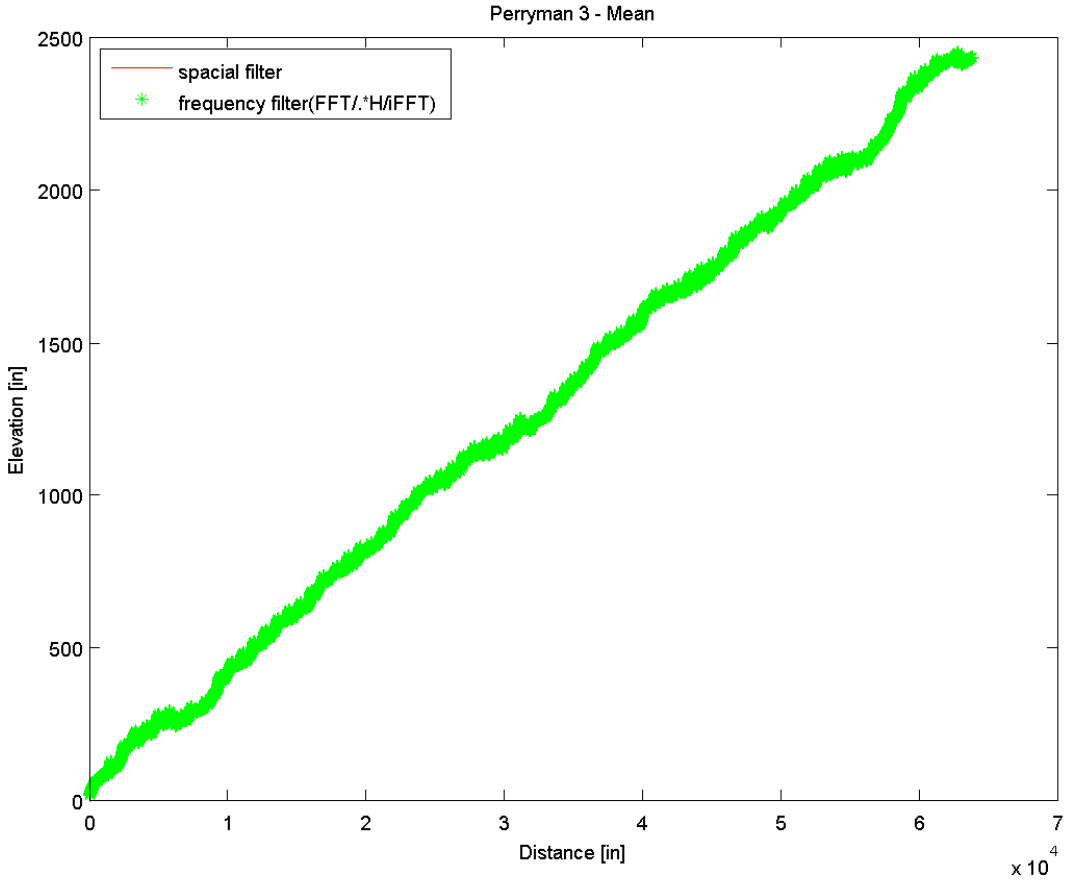


Figure 23 – Spacial versus Frequency Domain Mean Comparison

A zoomed version is shown in Figure 24 for the end of the terrain profile.

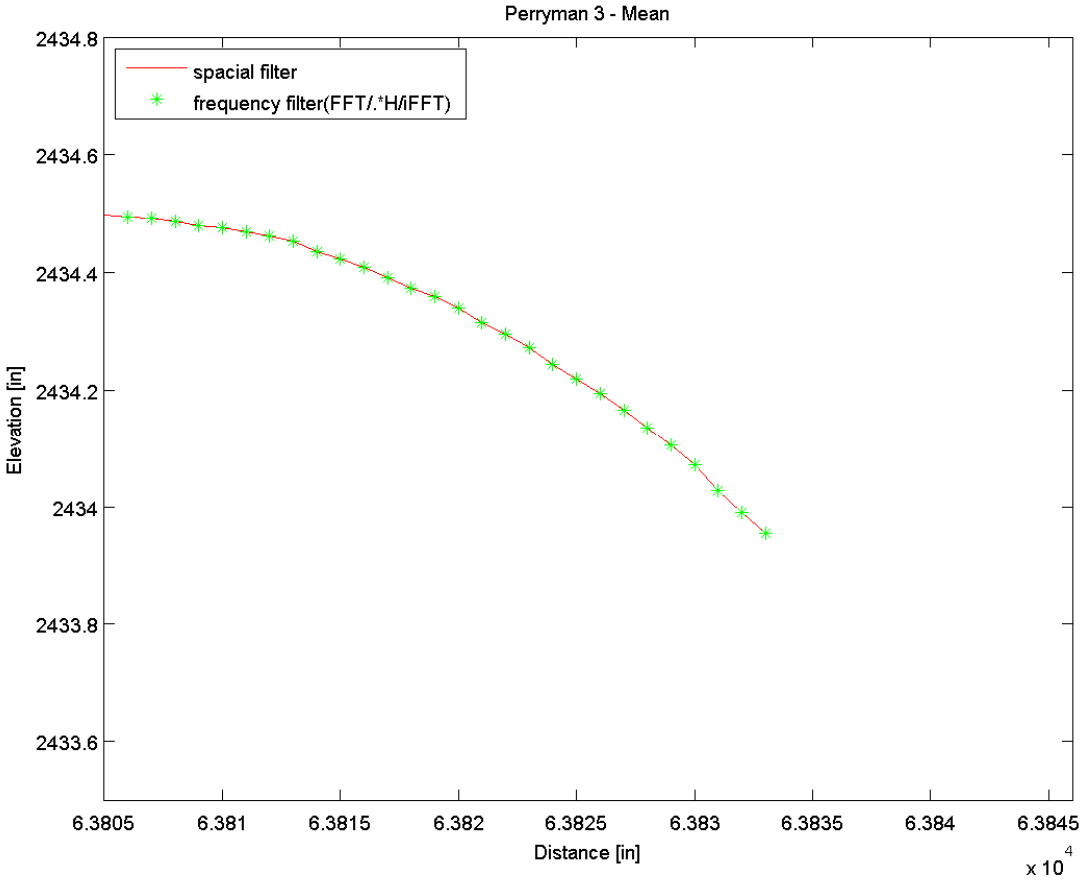


Figure 24 – Zoomed Mean Comparison

4.3.4 Spacial vs. Frequency Domain Detrend Comparison

Figure 25 shows the detrended terrain profile based on subtracting the trend from the frequency domain.

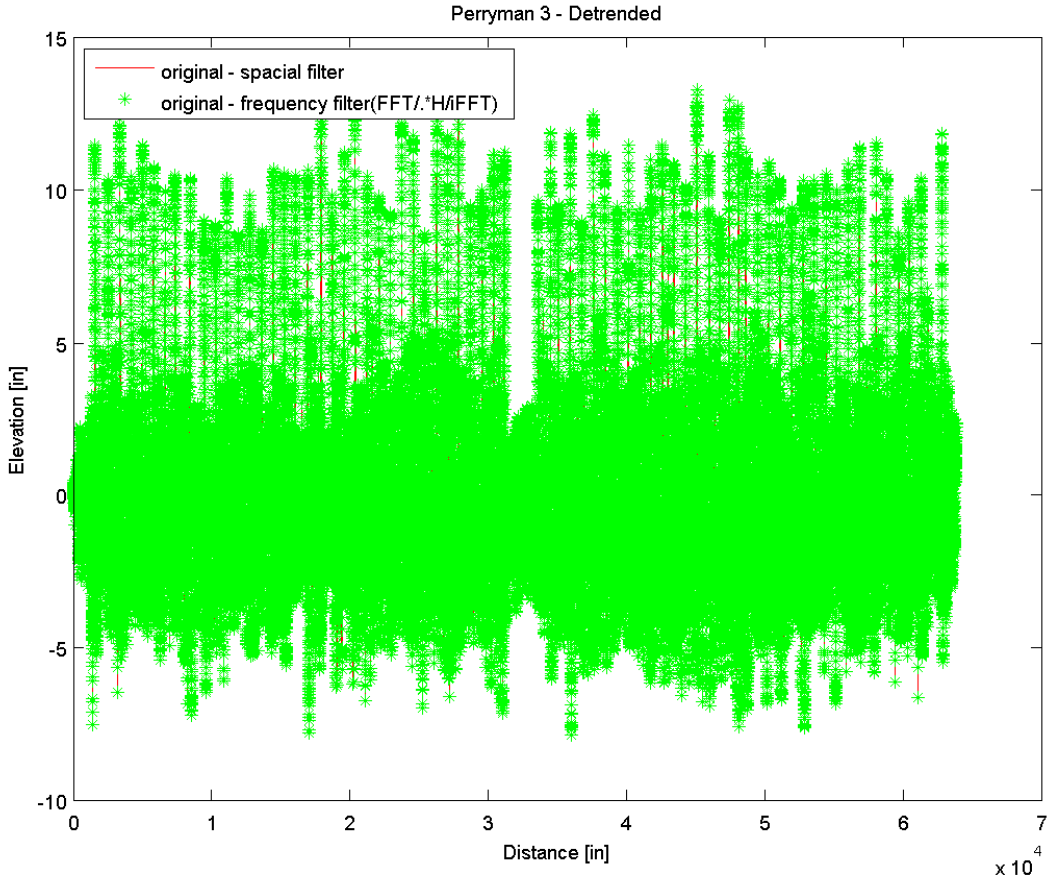


Figure 25 – Spacial versus Frequency Domain Detrended Comparison

A zoomed version of the end is shown in Figure 26. Note the agreement in results.

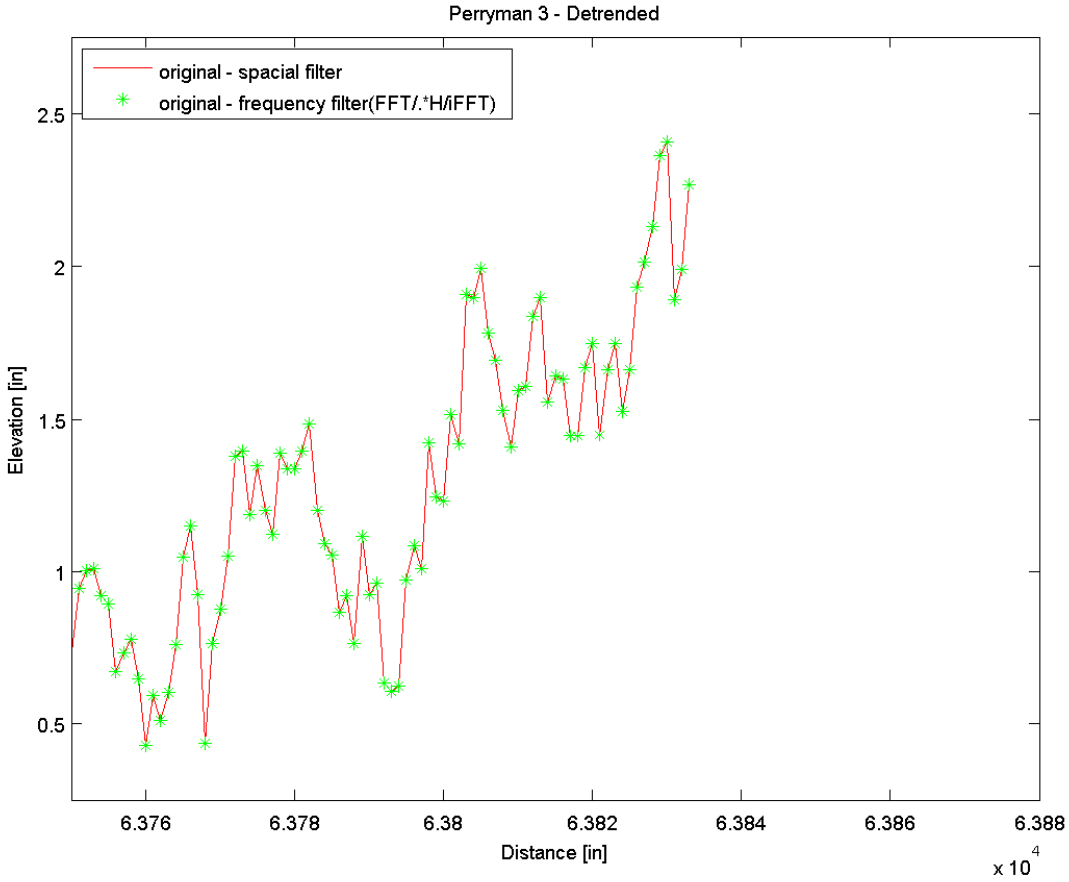


Figure 26 – Zoomed Detrended Comparison

4.3.5 Spacial vs. Frequency Domain Mean and Detrended Differences

Figure 27 shows the difference between the spacial and frequency domain for the mean and detrended profile. Note they are the same except for round off error.

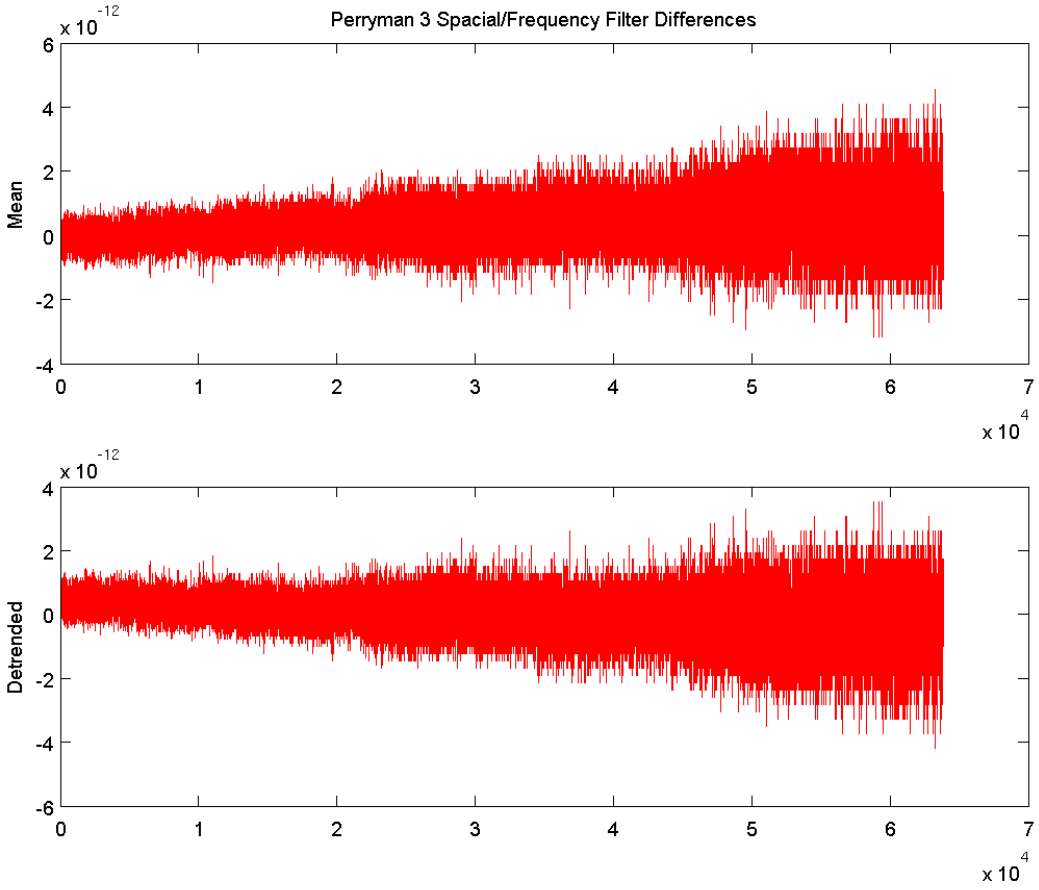


Figure 27 – Spacial versus Frequency Domain Mean/Detrended Differences

4.3.6 Equivalent Measures

Results output from the script in Section 4.2.8 for this terrain profile show numerical comparisons between the spacial domain RMS function, standard deviation of the detrended signal, square root of the variance of the detrended signal, and each side of Parseval's Theorem outlined in Section 3.3 to be equivalent.

```
ans =

spacial/freq results


ans =

rms


ans =

std(mydet,1)


ans =

sqrt(var(mydet,1))


ans =

sqrt( sum(mydet.^2)/length(mydet))


ans =

sqrt( sum( Pxx )*length(mydet)/NFFT)


ans =

    3.1228    3.1228    3.1228    3.1228    3.1228
```

Again, it is demonstrated (with a completely different terrain profile) that the computation of the Root-Mean-Square (RMS) Terrain Roughness Metric (Section 2.1) is valid in either the spacial or frequency domain.

4.4 Test Case Calculations

An example test case is included in Appendix A.5 where corresponding script functions and output can be compared for a simple signal

```
x = [1 3 -10 5 8 9 22 2 3 10 10 12 11 14 13 9 17]';
```

and filter

```
h = [1 2 3 8 3 2 1]';
```

to demonstrate how MATLAB functions (see Appendix A.2) perform.

REFERENCE

[1] Surface Roughness. ("http://en.wikipedia.org/wiki/Surface_roughness")

[2] Surface Texture. ("Explanation of Surface Characteristics"). Accretech Tokyo Seimitsu ("http://www.accretech.jp/english/pdf/measuring/sfexplain_e.pdf")

[3] Annex H Operational Terrain to Purchase Description for Joint Light Tactical Vehicle Version 3.0.2 8th March 2012. ("https://contracting.tacom.army.mil/majorsys/jltv_emd/Amendment%205/Attachment%201%20-%20PD%20Annex%20H%20v3.0.2.pdf") Approved for Public Release; Distribution Unlimited.

[4] Weiss, R. Terrain Microroughness and the Dynamic Response of Vehicles. Trans. of the 27th Conference of Army Mathematicians. West Point, NY, June 1981. ARO Report 82-1. DTIC ADA 120 945. Approved for Public Release; Distribution Unlimited.

[5] Simon and Roach. Measurement of the Cross-Country Terrain Environment. US Army Transportation Research Command. Shock, Vibration and Associated Environments, Part III, Bulletin No. 30, February 1962. DTIC AD 273515

[6] Ashmore, Hodges, and Prebeg. Terrain Severity Data Generation at Yuma Proving Ground. Nevada Automotive Test Center. Final Report No. 13491 Vol. 1, October 1989 – November 1990. DTIC ADA 231 656. Approved for Public Release; Distribution Unlimited.

[7] Ashmore, Hodges, and Prebeg. Terrain Severity Data Generation at Yuma Proving Ground. Nevada Automotive Test Center. Final Report No. 13491 Vol. 2, October 1989 – November 1990. DTIC ADA 231 657. Approved for Public Release; Distribution Unlimited.

[8] FM5-33 Terrain Analysis. (http://www.globalsecurity.org/military/library/policy/army/fm/5-33/)

[9] Steinwolf andConnon III. Limitations of the Fourier Transform for Describing Test Course Profiles. Sound and Vibration, February 2005. ("http://www.sandv.com/downloads/0502stei.pdf")

[10] Keenan. Effects of Terrain Power Spectral Density Shaping and Measurement Interval on a Vehicle Ride Simulation. Report SIT-DL-73-1646, February 1973. Stevens Institute of Technology. DTIC AD 756 497.

[11] Vehicle Test Course Severity - Test Operations Procedure (TOP) 1-1-010, 4 January 2012 (Pages 10-11). Automotive Directorate (CSTE-DTC-AT-AD), US Army Aberdeen Test Center, 400 Colleran Road, Aberdeen Proving Ground, MD 21005-5059. (https://vdls.atc.army.mil/). Approved for Public Release; Distribution Unlimited.

[12] Van Deusen, B.D. "A Statistical Technique for the Dynamic Analysis of Vehicles Traversing Rough Yielding and Non-Yielding Surfaces". NASA Report No. CR-659, March 1967.

[13] MATLAB. The Mathworks. (http://www.mathworks.com)

[14] Unpublished. Murphy, "A Method for Determining Terrain Surface Roughness". US Army Engineer Waterways Experiment Station, Geotechnical Laboratory, Vicksburg, MS. Sep 1984.

[15] Jacobs. "Correlation and Convolution", Class Notes for CMSC 426, Fall 2005 ("http://www.cs.umd.edu/~djacobs/CMSC426/Convolution.pdf")

[16] Smith. "The Scientist and Engineer's Guide to Digital Signal Processing", ("http://www.dspguide.com/pdfbook.htm")

[17] Parseval's Theorem. ("http://en.wikipedia.org/wiki/Parseval's_theorem")

[18] Discrete Fourier Transform ("http://en.wikipedia.org/wiki/Discrete_Fourier_transform")

[19] FFTW ("http://fftw.org").

APPENDIX A.1 – RMS (Exponential Weighted Filter)

```
%function [fdet,rms,fmean,xmean]=rms(x,N,A,lambda)
% calculate RMS exp.weighted average value of terrain
%
%inputs:
% x - array to filter
% N - number of samples vector eg. [0 1 2 3 4 5]
% A - sample spacing
% lambda - exp weighting constant
%
%returns:
% fdet - detrended terrain
% rms - rms value
% fmean - trend
% xmean - mean of fdet
%
function [fdet,rms,fmean,xmean]=rms(x,N,A,lambda);
% can extend input by length of filter so do not lose data
%x = [x(1)*zeros(1,length(N)-1), x', x(end)*zeros(1,length(N)-1)]';
n = length(x);
s = length(N);
e = (n+1) - s;
det1 = exp(-N*A/lambda);
d = sum(det1)*2;
fmean = x;
fdet = x;
for i = s:e,
  fmean(i) = sum(x(i:-1:i-s+1).*det1 + x(i:i+s-1).*det1)/d;
end
fdet(s:e) = x(s:e) - fmean(s:e);
xmean = mean(fdet(s:e));
fdet(s:e) = fdet(s:e)- xmean;
% for best unbiased estimator use std(x,0) to divide by N-1 instead of N
rms = std(fdet(s:e),1);
%
fmean = fmean(s:e);
fdet = fdet(s:e);
```

APPENDIX A.2 – MATLAB Function Definitions

dlmread (http://www.mathworks.com/help/matlab/ref/dlmread.html)
size (http://www.mathworks.com/help/matlab/ref/size.html)
nextpow2 (http://www.mathworks.com/help/matlab/ref/nextpow2.html)
conv (http://www.mathworks.com/help/matlab/ref/conv.html)
fft (http://www.mathworks.com/help/matlab/ref/fft.html)
ifft (http://www.mathworks.com/help/matlab/ref/ifft.html)

The functions `Y = fft(x)` and `y = ifft(X)` implement the transform and inverse transform pair given for vectors of length *N* by:

$$X(k) = \sum_{j=1}^{N} x(j)\omega_N^{(j-1)(k-1)}$$

$$x(j) = (1/N)\sum_{k=1}^{N} X(k)\omega_N^{-(j-1)(k-1)}$$

where

$$\omega_N = e^{(-2\pi i)/N}$$

is an *N*th root of unity.

var (http://www.mathworks.com/help/matlab/ref/var.html)
std (http://www.mathworks.com/help/matlab/ref/std.html)
conj (http://www.mathworks.com/help/matlab/ref/conj.html)
exp (http://www.mathworks.com/help/matlab/ref/exp.html)
linspace (http://www.mathworks.com/help/matlab/ref/linspace.html)
psd (http://www.mathworks.com/help/signal/ref/spectrum.html)

APPENDIX A.3 – Reference [15] – First 4 pages

Correlation and Convolution

Class Notes for CMSC 426, Fall 2005

David Jacobs

Introduction

Correlation and Convolution are basic operations that we will perform to extract information from images. They are in some sense the simplest operations that we can perform on an image, but they are extremely useful. Moreover, because they are simple, they can be analyzed and understood very well, and they are also easy to implement and can be computed very efficiently. Our main goal is to understand exactly what correlation and convolution do, and why they are useful. We will also touch on some of their interesting theoretical properties; though developing a full understanding of them would take more time than we have.

These operations have two key features: they are *shift-invariant*, and they are *linear*. Shift-invariant means that we perform the same operation at every point in the image. Linear means that this operation is linear, that is, we replace every pixel with a linear combination of its neighbors. These two properties make these operations very simple; it's simpler if we do the same thing everywhere, and linear operations are always the simplest ones.

We will first consider the easiest versions of these operations, and then generalize. We'll make things easier in a couple of ways. First, convolution and correlation are almost identical operations, but students seem to find convolution more confusing. So we will begin by only speaking of correlation, and then later describe convolution. Second, we will start out by discussing 1D images. We can think of a 1D image as just a single row of pixels. Sometimes things become much more complicated in 2D than 1D, but luckily, correlation and convolution do not change much with the dimension of the image, so understanding things in 1D will help a lot. Also, later we will find that in some cases it is enlightening to think of an image as a continuous function, but we will begin by considering an image as *discrete*, meaning as composed of a collection of pixels.

Notation

We will use uppercase letters such as I and J to denote an image. An image may be either 2D (as it is in real life) or 1D. We will use lowercase letters, like i and j to denote indices, or positions, in the image. When we index into an image, we will use the same conventions as Matlab. First, that means that the first element of an image is indicated by 1 (not 0, as in Java, say). So if I is a 1D image, $I(1)$ is its first element. Second, for 2D images we give first the row, then the column. So $I(3,6)$ is the pixel in the third row of the image, and the sixth column.

An Example

One of the simplest operations that we can perform with correlation is local averaging. As we will see, this is also an extremely useful operation. Let's consider a simple averaging operation, in which we replace every pixel in a 1D image by the average of that pixel and its two neighbors. Suppose we have an image *I* equal to:

5	4	2	3	7	4	6	5	3	6

Averaging is an operation that takes an image as input, and produces a new image as output. When we average the fourth pixel, for example, we replace the value 3 with the average of 2, 3, and 7. That is, if we call the new image that we produce *J* we can write: $J(4) = (I(3)+I(4)+I(5))/3 = (2+3+7)/3 = 4$. Or, for example, we also get: $J(3) = (I(2)+I(3)+I(4))/3 = (4+2+3)/3 = 3$. Notice that every pixel in the new image depends on the pixels in the old image. A possible error is to use $J(3)$ when calculating $J(4)$. Don't do this; $J(4)$ should only depend on $I(3)$, $I(4)$ and $I(5)$. Averaging like this is shift-invariant, because we perform the same operation at every pixel. Every new pixel is the average of itself and its two neighbors. Averaging is linear because every new pixel is a linear combination of the old pixels. This means that we scale the old pixels (in this case, we multiply all the neighboring pixels by 1/3) and add them up. This example illustrates another property of all correlation and convolution that we will consider. The output image at a pixel is based on only a small neighborhood of pixels around it in the input image. In this case the neighborhood contains only three pixels. Sometimes we will use slightly larger neighborhoods, but generally they will not be too big.

Boundaries: We still haven't fully described correlation, because we haven't said what to do at the boundaries of the image. What is $J(1)$? There is no pixel on its left to include in the average, ie., $I(0)$ is not defined. There are four common ways of dealing with this issue.

.	.	.	*0*	*0*	5	4	2	3	7	4	6	5	3	6	*0*	*0*	.	.	.

In the first method of handling boundaries, the original image is padded with zeros (in red italics).

The first way is to imagine that *I* is part of an infinitely long image which is zero everywhere except where we have specified. In that case, we have $I(0) = 0$, and we can say: $J(1) = (I(0) + I(1) + I(2))/3 = (0 + 5 + 4)/3 = 3$. Similarly, we have: $J(10) = (I(9)+I(10)+I(11))/3 = (3 + 6 + 0)/3 = 3$.

.	.	.	*5*	*5*	5	4	2	3	7	4	6	5	3	6	*6*	*6*	.	.	.

In the second method of handling boundaries, the original image is padded with the first and last values (in red italics).

The second way is to also imagine that *I* is part of an infinite image, but to extend it using the first and last pixels in the image. In our example, any pixel to the left of the first pixel in *I* would have the value 5, and any pixel to the right of the last pixel would have the value 6. So we would say: $J(1) = (I(0) + I(1) + I(2))/3 = (5 + 5 + 4)/3 = 4\ 2/3$, and $J(10) = (I(9)+I(10)+I(11))/3 = (3 + 6 + 6)/3 = 5$.

.	.	.	*3*	*6*	5	4	2	3	7	4	6	5	3	6	*5*	*4*	.	.	.

In the third method of handling boundaries, the original image is repeated cyclically (in red italics).

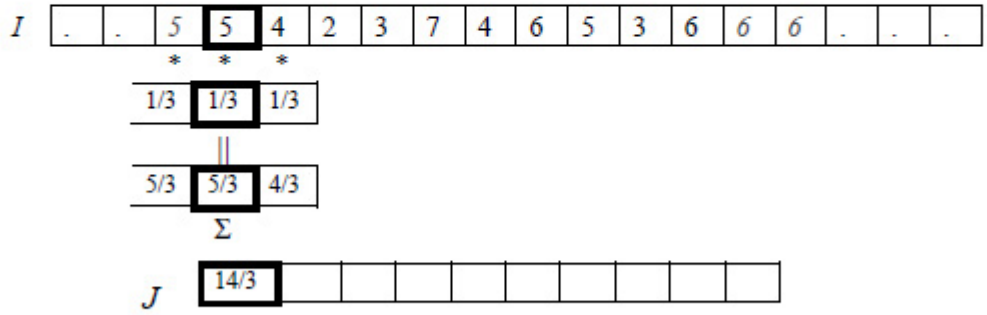
Third, we can imagine the image as being like a circle, so that the pixel values repeat over and over again. The pixel to the left of the first pixel, then, would be the last pixel in the image. That is, in our example, we would define *I(0)* to be *I(10)*. Then we would have $J(1) = (I(0) + I(1) + I(2))/3 = (I(10) + I(1) + I(2))/3 = (6 + 5 + 4)/3 = 5$, and $J(10) = (I(9)+I(10)+I(11))/3 = (I(9)+I(10)+I(1))/3 = (3 + 6 + 5)/3 = 4\ 2/3$.

Finally, we can simply say that the image is undefined beyond the values that we have been given. In that case, we cannot compute any average that uses these undefined values, so *J(1)* and *J(10)* will be undefined, and *J* will be smaller than *I*.

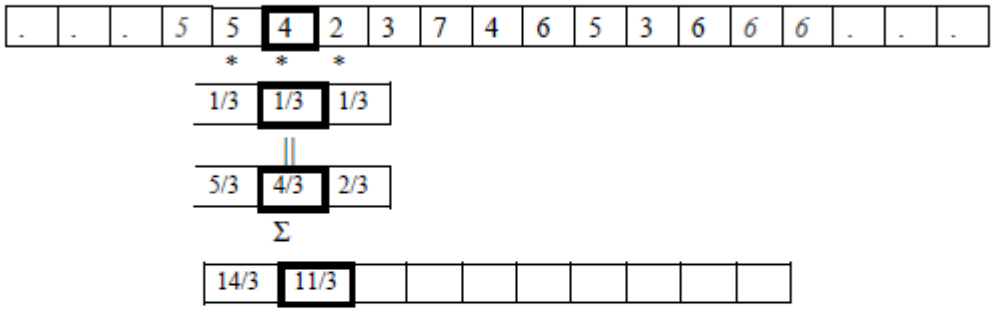
These four methods have different advantages and disadvantages. If we imagine that the image we are using is just a small window on the world, and we want to use values outside the boundary that are most similar to the values that we would have obtained if we'd taken a bigger picture, than the second approach probably makes the most sense. That is, if we had to guess at the value of *I(0),* even though we can't see it, the value we can see in *I(1)* is probably a pretty good guess. In this class, unless we explicitly state otherwise, you should use the second method for handling boundaries.

Correlation as a Sliding, Windowed Operation

We're now going to look at the same averaging operation in a slightly different way which is more graphical, and perhaps more intuitive to generalize. In averaging, for a specific pixel we multiply it and its neighbors by 1/3 each, and then add up the three resulting numbers. The numbers we multiply, (1/3, 1/3, 1/3) form a *filter*. This particular filter is called a *box* filter. We can think of it as a 1x3 structure that we slide along the image. At each position, we multiply each number of the filter by the image number that lies underneath it, and add these all up. The result is a new number corresponding to the pixel that is underneath the center of the filter. The figure below shows us producing *J(1)* in this way.



To produce the next number in the filtered image, we slide the filter over a pixel, and perform the same operation.



We continue doing this until we have produced every pixel in *J*. With this view of correlation, we can define a new averaging procedure by just defining a new filter. For example, suppose instead of averaging a pixel with its immediate neighbors, we want to average each pixel with immediate neighbors and their immediate neighbors. We can define a filter as (1/5, 1/5, 1/5, 1/5, 1/5). Then we perform the same operation as above, but using a filter that is five pixels wide. The first pixel in the resulting image will then be: *J(1) = (I(-1)/5 + I(0)/5 + I(1)/5 + I(2)/5 + I(3)/5) = 1+1+1+4/5 + 2/5 = 4 1/5.*

A Mathematical Definition for Correlation

It's helpful to write this all down more formally. Suppose *F* is a correlation filter. It will be convenient notationally to suppose that *F* has an odd number of elements, so we can suppose that as it shifts, its center is right on top of an element of *I*. So we say that *F* has *2N+1* elements, and that these are indexed from *-N* to *N*, so that the center element of *F* is *F(0)*. Then we can write:

$$F \circ I(x) = \sum_{i=-N}^{N} F(i) I(x+i)$$

where the circle denotes correlation. With this notation, we can define a simple box filter as:

$$F(i) = \begin{cases} 1/3 & \text{for} \quad i = -1,0,1 \\ 0 & \text{for} \quad i \neq -1,0,1 \end{cases}$$

Constructing an Filter from a Continuous Function

It is pretty intuitive what a reasonable averaging filter should look like. Now we want to start to consider more general strategies for constructing filters. It commonly occurs that we have in mind a continuous function that would make a good filter, and we want to come up with a discrete filter that approximates this continuous function. Some reasons for thinking of filters first as continuous functions will be given when we talk about the

APPENDIX A.4 – Plot Demonstration Script

```
%=====
%t1
close all
clear all
fig = 1
y=dlmread('MUNGL.FIL');
%y = dlmread('3courL.FIL');
x = y(:,2)*12; % convert feet to inches
tmp = size(x);
lenx = tmp(1);
% e-(N*A/lambda)
N = [0:1:120]'; % 1-side = 30 ft = 360 in @ 3 in samples = 120 (360/3)
A = 3; % 3 in samples
lambda = 10*12; % 10 ft * 12 = 120 in
% -N*A/lambda = -120*3/120 = -3 @ last sample
%[fdet,rms,fmean,xmean]=rmswes(x,N,A,lambda);
[fdet,rms,fmean,xmean]=rmswes(x,N,3,10*12); % 3 inch samples to 30 ft

figure(fig)
clf
fig = fig+1
tt1 = (0:lenx-1)'*(1/3);
plot(tt1,x,'r');
hold on
plot(tt1(121:end-120),fmean,'g');
plot(tt1(121:end-120),fdet,'b');
title('Munson Gravel');
%title('Perryman 3');
ylabel('Elevation [in]');
xlabel('Distance [in]');
legend('original','mean','detrended','Location','NorthWest');
axis([-50,15000,-200,1800])

print('-dpng','t1-mun');
%print('-dpng','t1-p3');

figure(fig)
clf
fig = fig+1
tt1 = (0:lenx-1)'*(1/3);
plot(tt1,x,'r');
hold on
plot(tt1(121:end-120),fmean,'g');
title('Munson Gravel');
%title('Perryman 3');
ylabel('Elevation [in]');
xlabel('Distance [in]');
legend('original','mean','Location','NorthWest');
axis([1.444e4,1.455e4,1732,1737])
print('-dpng','t1-zm1');

figure(fig)
clf
fig = fig+1
tt1 = (0:lenx-1)'*(1/3);
plot(tt1(121:end-120),fdet,'b');
title('Munson Gravel');
%title('Perryman 3');
ylabel('Elevation [in]');
xlabel('Distance [in]');
legend('detrended','Location','NorthWest');
axis([1.42e4,1.47e4,-4,3])
print('-dpng','t1-zm2');

%t2.m
%================================================
% create and plot filters
%
% 2-sided exponential
tmp = size(N);
lenN = tmp(1);
```

```
fexp = zeros(lenN*2-1,1); % overlap at zero
fexp(1:lenN) = exp(-N(length(N):-1:1)*A/lambda);
fexp(lenN:end) = exp(-N*A/lambda);
fexp(lenN) = 2; % double count zero position
fexp = fexp/sum(fexp);
tmp = size(fexp);
lenfexp = tmp(1);

% rect
rect = ones(lenN*2-1,1);
rect = rect/sum(rect);
tmp = size(rect);
lenrect = tmp(1);

figure(fig)
clf
fig = fig + 1

subplot(2,1,1),plot(fexp,'r')
ylabel('exp')
title('Spacial Windows - Normalized')
axis([-1,252,-0.01,0.03]);
xlabel('Distance [in]')
subplot(2,1,2),plot(rect,'g')
ylabel('rect');
xlabel('Distance [in]')
axis([-1,252,-0.01,0.01]);

print('-dpng','t2-tw');

%t3.m
%================================================
% plot filters FFT
%
lenx
NFFT = 2^nextpow2(lenx); % !!!use all with this so can mul in freq domain
%
lenfexp
lenrect

Ffexp = fft(fexp,NFFT)/NFFT; % !!!scale by len to get right mag (see sin test)
Frect = fft(rect,NFFT)/NFFT;
% Fs = 1/T
Fs=(1/3); % 4 samples in 12 in = 4/12 = 1/3

% linspace(x1,x2,N) = N pts between x1 and x2
f = Fs/2*linspace(0,1,NFFT/2+1);  %

figure(fig)
clf
fig = fig + 1

subplot(2,1,1),plot(f,2*abs(Ffexp(1:NFFT/2+1)),'r'); % !!! 2*abs /NFFT/2+1
title('Frequency Windows (FFT)');
ylabel('exp')
xlabel('Frequency [Hz]')
axis([0,0.05,0,4e-5]);
subplot(2,1,2),plot(f,2*abs(Frect(1:NFFT/2+1)),'g');
ylabel('rect')
xlabel('Frequency [Hz]')
axis([0,0.05,0,4e-5]);
print('-dpng','t3-fft');

%t4.m
%================================================
% plot cos test FFT
%
% MATLAB example
figure(fig)
clf
fig = fig + 1
Fs1 = 1000 % F=1/T =1/1e-3sec = 1/msec
L = 1000;
t= (0:1/Fs1:5000-1); % 5 sec @ Fs=1000
```

```
y = 0.3*sin(2*pi*35*t)+0.7 *sin(2*pi*50*t) + sin(2*pi*120*t);
tNFFT = 2^nextpow2(L);
tY = fft(y,tNFFT)/tNFFT;
tf = Fs1/2*linspace(0,1,tNFFT/2+1);  % one sided frequency

subplot(2,1,1),plot(1000*t(1:250),y(1:250))
title('FFT Example - 0.3cos(35Hz)+0.7cos(50Hz)+1.0cos(120Hz)');
ylabel('Signal')
xlabel('Time [msec]');
subplot(2,1,2),plot(tf,2*abs(tY(1:tNFFT/2+1)))
xlabel('Frequency [Hz]');
ylabel('FFT (2*|Y(f)|)');
print('-dpng','t4-cos');

%t5.m
%================================================
% plot filter inverse FFT
%
ffexp = NFFT*ifft(Ffexp,NFFT); % !!! scale by length
frect = NFFT*ifft(Frect,NFFT);

figure(fig)
clf
fig = fig + 1

subplot(2,1,1),plot(fexp-ffexp(1:lenfexp),'r'); % lenfexp = lenN*2-1
title('Spacial/Frequency(FFT/iFFT) Window Differences');
ylabel('exp');
subplot(2,1,2),plot(rect-frect(1:lenrect),'b');
ylabel('rect')
print('-dpng','t5-wd');

%t6.m
%================================================
% plot FFT x
%
Fx = fft(x,NFFT)/NFFT;
tt6=(0:lenx-1)*(1/3);
ft6 = Fs/2*linspace(0,1,NFFT/2+1);
%================================================
% plot inverse FFT x
%
Fcx = Fx;
fFcx = NFFT*ifft(Fcx,NFFT);

figure(fig)
clf
fig = fig + 1

plot(tt6,x,'r');
hold on
plot(tt6,fFcx(1:lenx),'g*'); % !!! NFFT -> true length
ylabel('Elevation [in]');
xlabel('Distance [in]');
title('Munson Gravel')
%title('Perryman 3')

'size x,fFcx'
size(x),size(fFcx)
plot(tt6,x-fFcx(1:lenx),'b');
legend('original','original(FFT/iFFT)','error','Location','NorthWest');
print('-dpng','t6-ter');

figure(fig)
clf
fig = fig + 1

plot(tt6,x,'r');
hold on
plot(tt6,fFcx(1:lenx),'g*'); % !!! NFFT -> true length
ylabel('Elevation [in]');
xlabel('Distance [in]');
title('Munson Gravel')
%title('Perryman 3')
```

```
'size x,fFcx'
size(x),size(fFcx)
legend('original','original(FFT/iFFT)','Location','NorthWest');
axis([1.45e4,1.4535e4,1733,1736]);
print('-dpng','t6-zm');

%t7.m
%================================================
% plot exp filter
%
H = Ffexp;
%H=conv(Ffexp,Frect);
tmp = size(H);
lenH = tmp(1);

%================================================
% plot inverse x, mean, mydet, detrended
%
Fcx = Fx.*H;
%Fcx = Fx.*(lenH*H).*Frect;
fFcx = NFFT*NFFT*ifft(Fcx,NFFT); % !!! scale by time length

figure(fig)
clf
fig = fig + 1

plot(fmean,'r')
hold on
plot(abs(fFcx(lenfexp:lenx+(lenfexp-1)-(lenfexp-1))),'g*'); % !!! proper indexes for y cutoffs (y
only when exp fully in orig y)
'size fmean,fFcx,lenx,lenfexp,lenN'
size(fmean),size(fFcx),lenx,lenfexp,lenN
title('Munson Gravel - Mean');
ylabel('Elevation [in]');
xlabel('Distance [in]');
legend('spacial filter','frequency filter(FFT/.*H/iFFT)','Location','NorthWest');
print('-dpng','t7-mean');

figure(fig)
clf
fig = fig + 1

plot(fmean,'r')
hold on
plot(abs(fFcx(lenfexp:lenx+(lenfexp-1)-(lenfexp-1))),'g*'); % !!! proper indexes for y cutoffs (y
only when exp fully in orig y)
'size fmean,fFcx,lenx,lenfexp,lenN'
size(fmean),size(fFcx),lenx,lenfexp,lenN
title('Munson Gravel - Mean');
%title('Perryman 3 - Mean');
ylabel('Elevation [in]');
legend('spacial filter','frequency filter(FFT/.*H/iFFT)','Location','NorthWest');
axis([4.333e4,4.3345e4,1735.25, 1735.45]);
print('-dpng','t7-zm');

%t8.m
%================================================
% plot detrended
% assumes lenexp is odd, take out mean from FFT/.*H/iFFT
mydet = x( (lenfexp+1)/2 : lenx - (lenfexp+1)/2 + 1 ) - abs( fFcx( lenfexp : lenx + (lenfexp-1) -
(lenfexp-1) ) );
mm=mean(mydet)
mydet = mydet - mm;

figure(fig)
clf
fig = fig + 1

plot(fdet,'r');
hold on
plot( mydet,'g*')
title('Munson Gravel - Detrended');
ylabel('Elevation [in]');
```

```
xlabel('Distance [in]');
legend('original - spacial filter','original - frequency
filter(FFT/.*H/iFFT)','Location','NorthWest');
print('-dpng','t8-det');

figure(fig)
clf
fig = fig + 1

plot(fdet,'r');
hold on
plot( mydet,'g*')
title('Munson Gravel - Detrended');
ylabel('Elevation [in]');
xlabel('Distance [in]');
legend('original - spacial filter','original - frequency
filter(FFT/.*H/iFFT)','Location','NorthWest');
axis([4.331e4,4.337e4,-0.4,0.8]);
print('-dpng','t8-zm');

%t9.m
%================================================
% plot mean error
%
figure(fig)
clf
fig = fig + 1

subplot(2,1,1),plot(fmean - abs(fFcx(lenfexp:lenx+(lenfexp-1)-(lenfexp-1))),'r'); % !!! proper
indexes for y cutoffs (y only when exp fully in orig y)
title('Munson Gravel Spacial/Frequency Filter Differences')
ylabel('Mean');

%================================================
% plot detrended error

subplot(2,1,2),plot(fdet-mydet,'r');
%title('Spacial/Frequency(iFFT) Munson Gravel Differences')
ylabel('Detrended');
print('-dpng','p9-diff');

%t10.m
%================================================
% PSD
% std(x,1) = sqrt(sum(x.^2)/length(x))
%
tmp = size(mydet);
lenmydet = tmp(1)

Fmydet = fft(mydet,NFFT)/NFFT;
Pxx = Fmydet.*conj(Fmydet);

!rm t10-d
diary t10-d
'spacial/freq restuls'
'rms'
'std(mydet,1)'
'sqrt(var(mydet,1))'
'sqrt( sum(mydet.^2)/length(mydet))'
'sqrt( sum( Pxx )*length(mydet)/NFFT)'

[rms,std(mydet,1),sqrt(var(mydet,1)),sqrt( sum(mydet.^2)/length(mydet)),sqrt( sum( Pxx
)*NFFT/length(mydet))]
%'Fs/2', sqrt( sum(2*Pxx)*Fs )
diary
stop

%EXAMPLE: Spectral analysis of a complex signal plus noise.
%      Fs = 1000;   t = 0:1/Fs:.296;
%      x = exp(1i*2*pi*200*t)+randn(size(t));
%      h = spectrum.periodogram;      % Create a periodogram spectral estimator.
%      psd(h,x,'Fs',Fs);              % Calculates and plots the two-sided PSD.

% EXAMPLE: Spectral analysis of a signal that contains a 200Hz cosine
```

```
%             % plus noise.
%             Fs = 1000;   t = 0:1/Fs:.296;
%             x = cos(2*pi*t*200)+randn(size(t));
%             h = spectrum.welch;                  % Create a Welch spectral estimator.
%             psd(h,x,'Fs',Fs);                    % Calculate and plot the PSD.
```

APPENDIX A.5 – Test Case Input/Output Example

```
%t11
% %%%!!! divid by NFFT (assumes zero mean, else need add mean/Pxx(0) etc.)
clear all
diary t11-d
%%%%%!!!!!NEED TO HAVE ZERO MEAN or changes things for RMS
x = [1 3 -10 5 8 9 22 2 3 10 10 12 11 14 13 9 17]';
x = x-mean(x);
tmp = size(x);
lenx = tmp(1);

x1 = [1 3 -10 5 8 9 22 2 3 10 10 12 11 14 13 9 17]';
x1 = [x1;1;0;-1;20;30];
x1 = x1 -mean(x1);

tmp = size(x1);
lenx1 = tmp(1);

%%%h
h = [1 2 3 8 3 2 1]';
tmp = size(h);
lenh = tmp(1);
'nextpow2'
lenx
nextpow2(lenx)
NFFT = 2^nextpow2(lenx)

%%%convolution
xh=conv(x,h,'full');

Fx = fft(x,NFFT)/NFFT;
Fh = fft(h,NFFT)/NFFT;
Fxh = Fx.*Fh;
iFxh = NFFT*NFFT*ifft(Fxh,NFFT);
iFxh = iFxh(1:lenx+lenh-1);
iFx = NFFT*ifft(Fx,NFFT);
iFh = NFFT*ifft(Fh,NFFT);
'x,h'
x,h
size(x),size(h)
'xh'
xh,iFxh
'size xh,iFxh'
size(xh),size(iFxh)
'fft(x,NFFT)/lenx'
Fx
'ifft(Fx,NFFT)*lenx'
iFx
'conj(Fx)'
conj(Fx)
'var(x),var(x,1)'
var(x),var(x,1)
'std(x),std(x,1)'
std(x), std(x,1)
'exp(-x)'
exp(-x)
'linspace(0,1,NFFT/2+1)'
NFFT
NFFT/2+1
linspace(0,1,NFFT/2+1)
'psd'

Fs = 1/3;  %T=3
h1=psd(spectrum.welch,0.5*sin(2*pi*Fs*[0:3:1000]),'Fs',Fs)
fprintf('%f %e\n',[h1.frequencies';h1.data']);
'======='
% PSD
Fx1 = fft(x1,NFFT)/NFFT;
'var x,x1'
a1=var(x,1)
b1=var(x1,1)
Fs = 1/3;
```

```
Pxx = Fx.*conj(Fx);
Px1 = Fx1.*conj(Fx1);
'sum x.^2 Pxx*NFFT; a1 sum Pxx*NFFT/lenx'
[sum(x.^2) sum(Pxx)*NFFT]
[a1 sum(Pxx)*NFFT/lenx]
%%%!!! divid by NFFT (assumes zero mean, else need add mean/Pxx(0) etc.)
'sum Px Px1 /NFFT'
c1=sum(Pxx)*NFFT/length(x)
d1=sum(Px1)*NFFT/length(x1)
%%%
r1 = ( a1*length(x) ) / ( c1*length(x) );
r2 = ( b1*length(x1) )/ ( d1*length(x1) );
'a1*len, c1*len^2 ratio, rep b1,d1'
fprintf('%f %f %f\n',[a1*length(x),c1*length(x)],r1);
fprintf('%f %f %f\n',[b1*length(x1),d1*length(x1)],r2);
'sqrt(a1), sqrt(c1/length(x))'
sqrt(a1),sqrt(c1)
'std(x,1)'
std(x,1)  % sqrt(a1*length(x))/sqrt(length(x))
'sqrt(var(x,1)*length(x)), std(x,1)*sqrt(length(x))'
sqrt(var(x,1)*length(x)),std(x,1)*sqrt(length(x))
a1*length(x),std(x,1)^2*(length(x))
diary



%
%===t11-d

ans =

nextpow2


lenx =

    17


ans =

     5


NFFT =

    32


ans =

x,h


x =

   -7.1765
   -5.1765
  -18.1765
   -3.1765
   -0.1765
    0.8235
   13.8235
   -6.1765
   -5.1765
    1.8235
    1.8235
    3.8235
    2.8235
    5.8235
    4.8235
    0.8235
    8.8235
```

```
h =

     1
     2
     3
     8
     3
     2
     1


ans =

    17     1


ans =

     7     1


ans =

xh


xh =

   -7.1765
  -19.5294
  -50.0588
 -112.4706
 -124.0000
 -184.3529
  -82.5294
  -28.5294
    5.4706
   82.4706
  -16.5294
  -18.5294
   16.4706
   26.4706
   59.4706
   67.4706
   89.4706
   85.6471
   62.0000
   88.5294
   32.9412
   18.4706
    8.8235


iFxh =

   -7.1765
  -19.5294
  -50.0588
 -112.4706
 -124.0000
 -184.3529
  -82.5294
  -28.5294
    5.4706
   82.4706
  -16.5294
  -18.5294
   16.4706
   26.4706
   59.4706
   67.4706
   89.4706
   85.6471
```

```
   62.0000
   88.5294
   32.9412
   18.4706
    8.8235


ans =

size xh,iFxh


ans =

    23     1


ans =

    23     1


ans =

fft(x,NFFT)/lenx


Fx =

   0.0000
  -1.6054 - 0.1955i
  -0.4515 + 0.8507i
  -0.9254 + 0.2054i
  -0.5466 + 1.1821i
   0.1333 + 1.1079i
   0.9118 + 0.4100i
   0.2412 - 0.3004i
  -0.0993 - 0.2500i
  -0.3327 + 0.4903i
   0.7963 + 0.1114i
   0.2423 - 0.4779i
   0.1605 - 1.0054i
  -1.1850 - 0.4994i
  -0.4037 + 0.1771i
  -0.5682 + 0.1820i
   0.0882
  -0.5682 - 0.1820i
  -0.4037 - 0.1771i
  -1.1850 + 0.4994i
   0.1605 + 1.0054i
   0.2423 + 0.4779i
   0.7963 - 0.1114i
  -0.3327 - 0.4903i
  -0.0993 + 0.2500i
   0.2412 + 0.3004i
   0.9118 - 0.4100i
   0.1333 - 1.1079i
  -0.5466 - 1.1821i
  -0.9254 - 0.2054i
  -0.4515 - 0.8507i
  -1.6054 + 0.1955i


ans =

ifft(Fx,NFFT)*lenx


iFx =

   -7.1765
   -5.1765
  -18.1765
   -3.1765
```

```
   -0.1765
    0.8235
   13.8235
   -6.1765
   -5.1765
    1.8235
    1.8235
    3.8235
    2.8235
    5.8235
    4.8235
    0.8235
    8.8235
    0.0000
         0
    0.0000
    0.0000
         0
   -0.0000
    0.0000
         0
   -0.0000
   -0.0000
    0.0000
   -0.0000
   -0.0000
    0.0000
         0


ans =

conj(Fx)


ans =

   0.0000
  -1.6054 + 0.1955i
  -0.4515 - 0.8507i
  -0.9254 - 0.2054i
  -0.5466 - 1.1821i
   0.1333 - 1.1079i
   0.9118 - 0.4100i
   0.2412 + 0.3004i
  -0.0993 + 0.2500i
  -0.3327 - 0.4903i
   0.7963 - 0.1114i
   0.2423 + 0.4779i
   0.1605 + 1.0054i
  -1.1850 + 0.4994i
  -0.4037 - 0.1771i
  -0.5682 - 0.1820i
   0.0882
  -0.5682 + 0.1820i
  -0.4037 + 0.1771i
  -1.1850 - 0.4994i
   0.1605 - 1.0054i
   0.2423 - 0.4779i
   0.7963 + 0.1114i
  -0.3327 + 0.4903i
  -0.0993 - 0.2500i
   0.2412 - 0.3004i
   0.9118 + 0.4100i
   0.1333 + 1.1079i
  -0.5466 + 1.1821i
  -0.9254 + 0.2054i
  -0.4515 + 0.8507i
  -1.6054 - 0.1955i


ans =

var(x),var(x,1)
```

```
ans =

   52.5294


ans =

   49.4394


ans =

std(x),std(x,1)


ans =

    7.2477


ans =

    7.0313


ans =

exp(-x)


ans =

   1.0e+07 *

    0.0001
    0.0000
    7.8332
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000
    0.0000


ans =

linspace(0,1,NFFT/2+1)


NFFT =

    32


ans =

    17


ans =

  Columns 1 through 12
```

```
         0    0.0625    0.1250    0.1875    0.2500    0.3125    0.3750    0.4375    0.5000
0.5625    0.6250    0.6875

  Columns 13 through 17

    0.7500    0.8125    0.8750    0.9375    1.0000


ans =

psd


h1 =

                  Name: 'Power Spectral Density'
                  Data: [129x1 double]
          SpectrumType: 'Onesided'
    NormalizedFrequency: false
                    Fs: 0.3333
           Frequencies: [129x1 double]
             ConfLevel: 'Not Specified'
           ConfInterval: []

0.000000 5.226231e-25
0.001302 9.483823e-25
0.002604 7.050443e-25
0.003906 4.228587e-25
0.005208 1.984093e-25
0.006510 6.876082e-26
0.007812 1.569868e-26
0.009115 1.860584e-27
0.010417 2.391157e-28
0.011719 2.802668e-28
0.013021 2.285926e-28
0.014323 2.048480e-28
0.015625 2.346595e-28
0.016927 3.117960e-28
0.018229 3.945803e-28
0.019531 4.274355e-28
0.020833 4.732021e-28
0.022135 7.948712e-28
0.023438 2.104163e-27
0.024740 5.810612e-27
0.026042 1.334442e-26
0.027344 2.441229e-26
0.028646 3.586796e-26
0.029948 4.296287e-26
0.031250 4.240851e-26
0.032552 3.464528e-26
0.033854 2.338243e-26
0.035156 1.299076e-26
0.036458 6.011988e-27
0.037760 2.513154e-27
0.039062 1.176740e-27
0.040365 7.297665e-28
0.041667 5.569431e-28
0.042969 4.908393e-28
0.044271 4.682606e-28
0.045573 4.532329e-28
0.046875 4.708755e-28
0.048177 5.515843e-28
0.049479 6.726700e-28
0.050781 8.231078e-28
0.052083 1.109477e-27
0.053385 1.771123e-27
0.054688 3.105492e-27
0.055990 5.335277e-27
0.057292 8.395125e-27
0.058594 1.172202e-26
0.059896 1.430514e-26
0.061198 1.513542e-26
0.062500 1.380611e-26
0.063802 1.079887e-26
```

0.065104 7.214590e-27
0.066406 4.147154e-27
0.067708 2.151817e-27
0.069010 1.147077e-27
0.070312 7.240367e-28
0.071615 5.296537e-28
0.072917 4.161614e-28
0.074219 3.547719e-28
0.075521 3.287241e-28
0.076823 3.259712e-28
0.078125 3.576679e-28
0.079427 4.289426e-28
0.080729 5.180134e-28
0.082031 6.253769e-28
0.083333 8.344877e-28
0.084635 1.289231e-27
0.085938 2.090314e-27
0.087240 3.191044e-27
0.088542 4.360914e-27
0.089844 5.248827e-27
0.091146 5.538457e-27
0.092448 5.117029e-27
0.093750 4.136936e-27
0.095052 2.923833e-27
0.096354 1.809939e-27
0.097656 1.004219e-27
0.098958 5.478430e-28
0.100260 3.493497e-28
0.101562 2.738669e-28
0.102865 2.308483e-28
0.104167 1.966835e-28
0.105469 1.747796e-28
0.106771 1.634736e-28
0.108073 1.669708e-28
0.109375 2.043268e-28
0.110677 2.827591e-28
0.111979 3.816106e-28
0.113281 4.964963e-28
0.114583 7.017416e-28
0.115885 1.141482e-27
0.117188 1.926689e-27
0.118490 3.011499e-27
0.119792 4.149922e-27
0.121094 4.983957e-27
0.122396 5.222663e-27
0.123698 4.794617e-27
0.125000 3.867756e-27
0.126302 2.739608e-27
0.127604 1.696213e-27
0.128906 9.218688e-28
0.130208 4.691263e-28
0.131510 2.750187e-28
0.132812 2.210799e-28
0.134115 2.098549e-28
0.135417 2.009576e-28
0.136719 1.903970e-28
0.138021 1.822989e-28
0.139323 1.894325e-28
0.140625 2.360866e-28
0.141927 3.378028e-28
0.143229 4.905343e-28
0.144531 7.012244e-28
0.145833 1.019448e-27
0.147135 1.505093e-27
0.148438 2.142128e-27
0.149740 2.785748e-27
0.151042 3.212215e-27
0.152344 3.249653e-27
0.153646 2.890991e-27
0.154948 2.293755e-27
0.156250 1.664718e-27
0.157552 1.135004e-27
0.158854 7.330027e-28
0.160156 4.449490e-28

```
0.161458 2.637662e-28
0.162760 1.785309e-28
0.164062 1.551579e-28
0.165365 1.539291e-28
0.166667 7.735081e-29

ans =

=======


ans =

var x,x1


a1 =

   49.4394


b1 =

   75.2417


ans =

sum x.^2 Pxx*NFFT; a1 sum Pxx*NFFT/lenx


ans =

  840.4706  840.4706


ans =

   49.4394   49.4394


ans =

sum Px Px1 /NFFT


c1 =

   49.4394


d1 =

   75.2417


ans =

a1*len, c1*len^2 ratio, rep b1,d1

840.470588 840.470588 1.000000
1655.318182 1655.318182 1.000000

ans =

sqrt(a1), sqrt(c1/length(x))


ans =

    7.0313


ans =
```

```
    7.0313

ans =

std(x,1)

ans =

    7.0313

ans =

sqrt(var(x,1)*length(x)), std(x,1)*sqrt(length(x))

ans =

   28.9909

ans =

   28.9909

ans =

  840.4706

ans =

  840.4706
```